Е. В. Андреева

Программирование — это так просто, программирование — это так сложно

Современный учебник программирования

Электронное издание

Москва
Издательство МЦНМО
2015

УДК 519.671
ББК 22.18
А65

Андреева Е. В.
Программирование — это так просто, программирование — это так сложно. Современный учебник программирования.
Электронное издание.
М.: МЦНМО, 2015.
184 с.
ISBN 978-5-4439-2313-0

Книга представляет собой практический курс для обучения программированию, основную часть которого составляет подборка около 200 задач. В ней делается попытка показать, как обучить программированию в школе за 16 уроков. Рассмотрены все алгоритмы из перечня, входящего в «Требования к уровню подготовки выпускников» согласно нормативным документам ЕГЭ.

Большинство приведенных задач предполагают проверку их решений на системе тестов. Автоматическая проверка решений задач будет организована на сайте informatics.mccme.ru.

Книга предназначена для учителей информатики и старшеклассников, изучающих информатику на профильном уровне или готовящихся к ЕГЭ по информатике. Пособие может быть использовано на кружковых и факультативных занятиях, а также в школах с углубленным изучением математики и информатики при изучении программирования учениками 8—9 классов.

Подготовлено на основе книги:
Андреева Е. В. Программирование — это так просто, программирование — это так сложно. Современный учебник программирования.— М.: МЦНМО, 2015. — 2 изд., испр. и доп. — ISBN 978-5-4439-0259-3

Издательство Московского центра
непрерывного математического образования
119002, Москва, Большой Власьевский пер., 11,
тел. (499)241–08–04.
http://www.mccme.ru

ISBN 978-5-4439-2313-0

Оглавление

Предисловие

Введение курса информатики в школах нашей страны фактически начиналось с преподавания программирования. В то время даже был провозглашен лозунг: «Программирование — это вторая грамотность». Заметим, что компьютеры в школах тогда практически отсутствовали. А та техника, которой оснащались школы в конце 80-х — начале 90-х годов прошлого века, практическую составляющую курса информатики все равно невольно сводила к программированию.

Одновременно с революционным развитием аппаратного и программного обеспечения и оснащением современной компьютерной техникой учебных заведений курс информатики претерпел существенные изменения. Основное внимание в большинстве школ стало уделяться освоению современных информационных технологий. Эти тенденции отражены и в новом Стандарте по информатике, в котором собственно обучению программированию отводится очень мало времени. Но, как заметил автор школьных учебников по информатике А. Г. Гейн, «...очевидно, что именно алгоритмизация с самого начала вытянула на школьную арену курс информатики и ныне во многих реально существующих курсах информатики позволяет уйти от умных, но пустоватых разговоров к конкретному делу (не случайно альтернативой алгоритмизации нередко выступает обучение информационным технологиям — учить детей тому и другому многим представляется невозможным, ибо освоение реального дела требует значительных затрат и труда, и времени)».

Кого следует учить программированию

В рамках часов, отводимых Примерной программой в базовом курсе информатики на алгоритмизацию и программирование, овладение даже основами программирования на современных алгоритмических языках представляется невозможным. Тем не менее, контингент школьников, у которых интерес именно к изучению, а не знакомству с программированием высок, несомненно, существует. В первую очередь это учащиеся физико-математических школ, гимназий, лицеев и гимназических классов общеобразовательных школ. У большинства из них есть как мотивация, так и способности к освоению программирования. Учебные планы подобных образовательных учреждений, в которых на освоение информатики и информационных технологий отводится не менее часа начиная с 5-го класса, не

менее двух — с 8-го и до четырех часов в 10—11 классах, также играют положительную роль. Мотивация есть и у учителя — ведь большинство современных олимпиад по информатике являются по своей сути олимпиадами по программированию, а по успехам учеников в олимпиадах зачастую судят о квалификации учителя, хотя в случае с информатикой это далеко не бесспорно. Кроме того, любовь к программированию многие учителя информатики принесли из своей профессиональной деятельности, и, конечно же, им хочется передать эту любовь и своим ученикам.

Таким образом, очевидно, что существуют школы, в которых могут и хотят учить школьников программированию, и естественно возникают вопросы — в каком возрасте начинать обучение и какой язык программирования лучше использовать? Ответ на первый вопрос не очень сложный. Если в 5—6 классах школьники уверенно освоили большинство современных информационных технологий: текстовый процессор (редактор), программу для создания презентаций, иногда — электронную таблицу (здесь намеренно не упоминается графический редактор, так как его освоение в простейшей форме сейчас происходит в основном уже в начальной школе, а профессиональные редакторы типа PhotoShop заслуживают детального рассмотрения либо в более старших классах, либо на факультативах), то уже в 7—8 классах можно попытаться построить курс информатики так, что его стержнем окажется именно изучение программирования. При этом теоретическая часть программы базового курса информатики не только не будет проигнорирована, но и, наоборот, будет освоена учащимися на более глубоком, практическом уровне.

В основе предлагаемого курса программирования лежит многолетний опыт работы автора в физико-математической школе-интернате им. А. Н. Колмогорова (СУНЦ МГУ) и с учениками 8—9 классов школы «Интеллектуал» г. Москвы. Изучение языка Logo или аналогичного ему в младших классах средней школы, конечно же, является чрезвычайно полезной пропедевтикой данного курса, однако непосредственно опираться на него мы не будем. Начинать преподавание программирования можно как в 8-м (иногда 7-м), так и в 9—10 классах, при этом незначительно меняются решаемые на уроках задачи, которые должны быть адаптированы к уровню математической подготовки учащихся. Отметим, что полноценные занятия можно проводить лишь тогда, когда на уроки информатики отводится не менее двух (спаренных) учебных часов в неделю. В противном случае изучение программирования лучше проводить в рамках факультатива.

Выбираем язык программирования

Перейдем к обсуждению выбора языка программирования. Здесь, на наш взгляд, категоричной рекомендации быть не может. Единственное условие, которое должно выполняться,—это то, что не только язык, но и выбранная среда программирования должны быть одними из реально используемых в современной практике, в том числе и профессионалами. Обратимся к мировому опыту как обучения программированию, так и профессионального программирования.

Basic, Quick Basic и Visual Basic

Долгие годы считалось, что язык программирования Basic является с методической точки зрения непригодным для обучения даже началам программирования будущих профессионалов, так как программирование с GoTo приводит к формированию плохого стиля, исправить который в дальнейшем очень сложно. Однако эволюция языка, начиная с Quick Basic и заканчивая Visual Basic, привела к тому, что сам язык стал мало отличаться, например, от языков Pascal и Delphi соответственно, и категоричные высказывания о непригодности его использования в учебных целях вряд ли можно считать корректными. Тем не менее, при выборе этого языка следует учитывать, что на многих олимпиадах высокого уровня по информатике и программированию в списке допустимых языков программирования Basic либо отсутствует вовсе (международная олимпиада школьников по информатике, студенческий чемпионат мира по программированию), либо присутствует в качестве одной из версий, зачастую весьма отличной от той, что изучалась в школе (так, в школе часто изучают Quick Basic, а на всероссийской олимпиаде в последние годы разрешенным является уже Visual Basic, причем написание корректных консольных приложений на данном языке даже у знакомых с ним школьников вызывает большие трудности).

Если же обратиться к статистике использования тех или иных языков программирования на международном рынке труда, то Visual Basic в настоящее время занимает почетное второе место, уступая лишь Visual C++. Однако это далеко не так у нас в стране, где аналогичную Visual Basic программистскую нишу прочно занимает Delphi. Единственным неоспоримым аргументом в пользу выбора Visual Basic в качестве базового для изучения программирования вообще является то, что именно этот язык используется для написания макрокоманд в современных офисных приложениях. И если изучение, например, электронных таблиц ведется на уровне, предполагающем свободное

владение этим языком, то выбор Visual Basic для освоения алгоритмизации и программирования может быть оправданным. В данном случае оказывается возможным построить интегрированный курс одновременного освоения как информационных технологий, так и программирования. Многие учителя отмечают также легкость «быстрого старта» при знакомстве школьников с данным языком. Но мой опыт работы показывает, что школьники, способные к овладению программированием вообще, очень быстро преодолевают сложности работы с любой средой программирования и легко осваивают формальные правила записи программ на изучаемом языке, в частности описание переменных с корректным указанием их типов.

Итак, большинство аргументов «за» и «против» языка Basic рассмотрены. Перейдем к анализу языков группы Pascal и сравнению их с «Си-подобными» языками.

Pascal или C (C++)

Напомним, что язык Pascal был создан в начале 70-х годов прошлого столетия выдающимся специалистом в области computer science Никлаусом Виртом именно как язык для изучения программирования. Основой для построения синтаксических конструкций этого языка послужил широко распространенный в то время Algol (**algo**rithmical **l**anguage). Вирт продолжил свою работу над созданием методически обоснованного языка программирования, предложив общественности сначала язык modula-2, а затем объектно-ориентированный Oberon. Однако последние два языка не получили сколь-либо широкого распространения в отличие от языка Pascal, чрезвычайной популярности которого способствовало развитие семейства компиляторов фирмы Borland, начиная от Turbo Pascal и заканчивая Delphi. Не все новшества, привнесенные специалистами Borland в классический Pascal, кажутся Вирту оправданными, тем не менее, и они, в том числе, привели к тому, что Pascal долгие годы занимал одно из ведущих мест среди профессиональных языков разработки различных приложений, а проект Delphi придал ему новое дыхание.

С методической точки зрения Pascal действительно хорошо подходит на роль учебного языка. Он позволяет познакомиться с большинством понятий современного программирования, освоить как различные типы, так и структуры данных. Программы на Pascal легко читаются, а один из важнейших принципов современного программирования — «удобочитаемость более важна, чем краткость кода» (конечно, если это не приводит к замедлению работы программы более чем в два-три раза), ведь над современными программны-

ми комплексами трудятся целые коллективы программистов, и им необходимо быстро ориентироваться в коде друг друга. Не случайно при описании различных алгоритмов в большинстве учебной литературы, в том числе и западной, используется именно Pascal или схожий с ним псевдокод. Удобочитаемость Pascal весьма кстати и учителю при проверке программ, написанных школьниками. Кроме того, синтаксис языка устроен так, что своей строгостью фактически вынуждает писать правильные программы. Это выгодно отличает Pascal, например, от языка C (C++), который, давая программисту широкие возможности, требует от него знания многих нюансов, зачастую упускаемых из вида начинающими программистами. Чего стоит, например, конструкция

```
if (a = b),
```

которая, являясь синтаксически корректной, имеет в C весьма далекую от интуитивного смысла семантику[1] (переменной `a` присваивается значение переменной `b`, затем значение `a` сравнивается с нулем). Или расссмотрим логическое выражение вида

```
a < b < c.
```

Оно является некорректным для числовых и символьных типов языка Pascal и не будет пропущено компилятором, но вполне воспринимается компиляторами с языка C (C++), однако имеет смысл, весьма отличный от двойного неравенства в математике: сначала выполняется сравнение `a` и `b`, результатом которого оказывается либо 1 (истина), либо 0 (ложь), а затем уже это число (0 или 1) сравнивается с `c`. Ситуацию, в которой такое сравнение имеет смысл, придумать практически невозможно. Завершить обзор подобных нюансов (сами же нюансы на этом вовсе не заканчиваются) хочется семантикой оператора `switch` в языке C (C++), которая также зачастую приводит к написанию некорректных программ начинающими программистами, так как только оператор `break` в конце описания каждого из вариантов переключателя позволяет придать данной конструкции тот смысл, в котором она в большинстве случаев используется (только в этом случае она превращается в аналог оператора `case` из других языков программирования).

Не случайно на факультете вычислительной математики и кибернетики МГУ им. М. В. Ломоносова в большинстве групп до последнего

[1] Синтаксис языка определяет правила записи на нем программ, а семантика — что означает та или иная языковая конструкция (например, как должен выполняться тот или иной оператор).

времени курс программирования начинался именно с изучения языка Pascal, причем в его классической версии... Казалось бы, на этом вопрос о выборе языка программирования для современной школы можно считать решенным. Однако не все так просто.

Если обратиться к опыту стран юго-восточной Азии, а именно в них широко ведется преподавание школьникам программирования, то там уже долгие годы обучение старшеклассников проводится на базе языка C++. А, например, в Южной Корее мне удалось наблюдать, как ученики 6—7 классов (вернее, классов, которые соответствуют нашим 6—7, так как корейские школы имеют три ступени, в каждой из которых классы нумеруются от 1) уверенно программировали на Java. Правда, происходило это в воскресной школе дополнительного образования при одном из местных университетов, но подобные формы обучения вполне соответствуют нашим кружкам или факультативам. Кроме того, насколько мне известно, еще в 2001 г. болгарские специалисты в области преподавания информатики в средних и высших учебных заведениях подготовили новые учебники, предназначенные для массового перевода школ страны на преподавание языка C++ вместо Pascal. В нашей же стране примерно в это же время осуществлялся массовый отход от преподавания программирования вообще. Выбор профессионалов в последние годы также лежит между C++ и Java. Более того, все так называемые скриптовые языки и языки web-программирования имеют C-подобный синтаксис. А для операционных систем UNIX-класса именно язык C является фактически родным. Наступление на Pascal ведется и на олимпиадном фронте: его использованию на студенческих чемпионатах мира, видимо, пришел конец (а разрешенными останутся как раз только C++ и Java). Все это вновь заставляет задуматься о выборе языка программирования.

В нашей стране также есть школы, в которых весьма успешно ведется преподавание программирования на языке C (или C++). В основном это физико-математические школы (например, школа № 30 г. Санкт-Петербурга). Большую роль здесь играет как контингент учащихся, так и личность учителя. В ряде школ C изучают как второй после Pascal язык (например, в гимназии № 1514 г. Москвы).

Мы также пытались вести преподавание программирования на языке C++, по крайней мере для тех из наших учеников, которые к 10 классу уверенно программировали на языке Pascal. К сожалению, этот опыт оказался не слишком удачным. Помимо уже упомянутых выше сложностей, возникающих при написании программ на C++, особенно на начальном этапе его освоения, мы столкнулись со сложностями технического порядка: настройка среды програм-

мирования требует некоторого навыка и зачастую нетривиальна в условиях школьной локальной сети, а тратить на нее драгоценное время урока — непозволительная роскошь. Сообщения компилятора о различного рода ошибках не столь информативны, как в средах программирования для языка Pascal фирмы Borland.

Pascal или Python

В первом издании язык программирования Python не обсуждался. За прошедшее время автор приобрел опыт обучения школьников 7—9 классов программированию именно на этом языке. И этот опыт оказался крайне удачным, даже по сравнению с Pascal. Порог вхождения в него сопоставим с Quick Basic, а возможности и распространенность в профессиональной среде существенно выше.

Главным недостатком выбора языка Python в качестве первого учебного языка программирования коллеги называют динамическую типизацию и сокрытие информации о представлении данных в компьютере. Так, для целых чисел в Python используется и стандартная компьютерная арифметика, и так называемая «длинная», при этом программисту не надо беспокоиться о выборе одной из них, как в JAVA, — интерпретатор сам примет правильное решение. Но этот так называемый недостаток для большинства программистов на Python, наоборот, кажется достоинством. Реальный же недостаток, который не позволяет автору полностью вести обучение программированию только на Python, — это быстродействие существующих интерпретаторов. Из-за невысокой скорости выполнения программ на Python решение, например, олимпиадных задач по информатике начиная с какого-то уровня, становится невозможным.

Тем не менее Python хорошо подходит в качестве единственного языка программирования тем школам, где обучение программированию ведется ознакомительно или в рамках подготовки к ЕГЭ по информатике. И следующие материалы автора по обучению программированию, несомненно, будут уже на этом языке. В настоящее время на сайте `informatics.msk.ru` выложен авторский курс учителя школы № 179 г. Москвы Кириенко Д. П., включающий в себя начальное описание языка Python 3 и множество задач как для начинающих, так и для сильных учащихся.

Версии Pascal

В настоящее время в российских школах используется несколько различных версий языка Pascal. Долгое время наиболее распространенной была среда программирования Borland Pascal 7.0 (BP 7.0).

BP 7.0 был выпущен в 1993 г., то есть этой замечательной среде программирования уже больше 20 лет. Фирмой Borland она давно не поддерживается, в результате чего она все меньше совместима с современными высокоскоростными компьютерами и операционными системами. Ограничения, налагаемые языком Borland Pascal на размер используемой памяти, возникшие в результате ориентации на DOS-модель памяти, не выдерживают никакой критики (64 Кб на все глобальные переменные, столько же на размер стека и 200—400 Кб на динамические переменные, редко используемые в школьных курсах). Ну и, наконец, невозможность простого и быстрого создания оконных приложений с различными элементами интерфейса, отвечающих современным требованиям.

Все эти проблемы были решены при создании среды Delphi (для работы в школе можно рекомендовать любую версию начиная с 6-й), однако коммерческий характер этой среды постепенно делает нереальным ее использование в российских школах. На международном уровне, в том числе на международной олимпиаде по информатике, де факто стандартом стал компилятор Free Pascal и соответствующие ему среды. Так, бесплатным аналогом среды Delphi является Lazarus. И практически все относящееся в данном издании к Delphi применимо также и Free Pascal. Ознакомиться со средой Lazarus можно на официальном сайте `www.lazarus.freepascal.org`.

Спешим успокоить тех, кому пока морально или технически тяжело перейти на подобный объектно-ориентированный язык с визуальной средой программирования: наш курс не будет опираться на особенности такой среды в частности и визуального программирования вообще. Рассматриваться будут лишь задачи, решаемые в рамках так называемых *консольных* (в отличие от *оконных*) приложений, требующие предварительного нахождения *алгоритма* решения. Особенности языка Object Pascal (то есть языка Delphi) будут лишь изредка упоминаться в сравнении с аналогичными в языке Borland Pascal.

Другой альтернативой для многих стала российская версия среды и компилятора с языка Pascal — PascalABC. В нем подкупает русскоязычный интерфейс, удобство работы в операционной системе Windows, простота в создании графических приложений и многое другое. Тем не менее данная среда лишь недавно стала предлагаться участникам официальных олимпиад по программированию, компилятор с этого языка не поддерживает другие версии Pascal, а разработчики языка пошли по пути его совершенствования, что делает программы, написанные для PascalABC, непереносимыми для других компиляторов. Наконец, данная среда совсем не известна в мире, что

в каком-то смысле делает ее вещью в себе. Поэтому мы не будем в данном издании разбирать возможности этой версии языка Pascal.

Два подхода к изучению программирования

Пусть выбор языка и среды программирования сделан. Рассмотрим теперь два подхода к изучению языка программирования: *формальный* и «программирование по образцу». Первый основан на формальном (строгом) описании конструкций языка программирования (*синтаксиса* языка и его *семантики*) тем или иным способом (с помощью синтаксических диаграмм, мета-языка или формального словесного описания) и использовании при решении задач только изученных, а следовательно, понятных элементов языка. При втором подходе школьникам сначала выдаются готовые программы, рассказывается, что именно они делают, и предлагается написать похожую программу или изменить имеющуюся, не объясняя до конца ряд «технических» или несущественных с точки зрения учителя для решения задачи деталей. При этом говорится: «Точный смысл соответствующих конструкций вы узнаете позднее, а пока поступайте аналогичным образом». Второй подход дает возможность так называемого «быстрого старта», но создает опасность получить полуграмотных пользователей среды программирования, то есть людей, которые используют в своей практике достаточно сложные конструкции, но не могут четко объяснить, почему в том или ином случае нужно применять именно их и как они работают. В результате рано или поздно такие «программисты» сталкиваются с ошибками, исправить которые они просто не в состоянии, — им не хватает знаний.

В своей практике при работе с десятиклассниками мы используем в основном первый, формальный подход. При этом некоторыми неформальными умениями эти школьники чаще всего уже обладают. На наш взгляд, одна из задач школьной информатики — научить именно формальному подходу, в частности при применении различных определений. И формальное изучение языка программирования этому немало способствует. Но и без хороших примеров (образцов) при обучении программированию школьников не обойтись. И чем младше ученики, тем больше примеров необходимо приводить при описании языка (иногда даже заменяя ими строгое определение). Другое дело, что, на наш взгляд, следует добиваться того, чтобы в результате обсуждения примера все его детали оказались понятны школьникам (обязательно нужно объяснить, как и почему это работает, в том числе опираясь на уже изученный формальный материал).

В этом случае сильные ученики получат возможность понять все досконально и смогут использовать полученные знания в дальнейшем, а средние — приобретут конкретные навыки.

Методика обучения программированию

Итак, все вводные замечания сделаны. Перейдем к собственно методике обучения, неотъемлемой частью которой является и сам изучаемый материал. В этой книге под словом *урок* будет подразумеваться логически законченная единица учебного материала (аналогично английскому слову *lesson*), а количество учебных часов, отводимых на изучение того или иного урока, может варьироваться. При кружковой работе вообще удобно предоставить каждому школьнику возможность двигаться в своем собственном темпе. За рамками рассмотрения останутся основы использования выбранной среды программирования, которые на первом уроке вкратце должны быть продемонстрированы: как создать новый файл для редактирования текста программы, как его сохранить и в дальнейшем открыть, как запустить программу на компиляцию и выполнение и как увидеть результат работы программы.

Многолетний опыт работы в школе и анализ контрольных работ слушателей курса повышения квалификации «Методика преподавания программирования на уроках информатики» педагогического университета «Первое сентября» показывают, что учителя на уроках обучения программированию сталкиваются не только с проблемой подбора задач, но и со сложностями их проверки. Даже опытный учитель не в состоянии во время урока качественно оценить программы всех своих учеников путем анализа текстов этих программ. Зачастую проверка заключается в запуске программы ученика и проверке результатов ее работы на 1—2 (часто очевидных) тестах.

В последнее время в нашей стране появилось несколько систем автоматической проверки программ, хотя они еще и не стали общедоступными. Тем не менее, наличие такой системы не снимает проблемы методики тестирования. Человеку, который не является специалистом в этой области (а таковыми специалистами невольно стали педагоги, активно занимающиеся подготовкой своих учеников к олимпиадам по программированию или проведением подобных олимпиад), сложно продумать и подготовить исчерпывающую систему тестов даже для стандартных задач.

Вашему вниманию предлагается практический курс для обучения программированию, основную часть которого составляет подборка около 200 задач, решение практически каждой из которых полезно

при изучении той или иной темы. Именно это отличает данную публикацию от других учебников и задачников по программированию, где приводится либо недостаточно задач, либо много однотипных. Фактически делается попытка показать, как обучить программированию в школе за 16 уроков :-). Конечно, на самом деле это невозможно. Однако количество различных тем для изучения действительно не превышает 16. А наличие автоматизированной системы проверки позволяет вынести почти всю практическую часть курса на дом (ученики с удовольствием самостоятельно решают задачи, если могут сразу видеть результат их проверки).

Большинство представленных задач предполагают проверку их решений на системе тестов. Такая проверка организована на сайте `informatics.msk.ru`. Специфика задач, решение которых проверяется автоматически, такова, что их условия должны быть полностью формализованы. А именно, подробно описывается диапазон всех вводимых значений, а также формат ввода и вывода. Для лучшего понимания и исключения неверного толкования приводятся примеры входных и выходных данных. Именно так и оформлены большинство предлагаемых задач.

Благодарности

Спасибо всем моим ученикам, для которых эта книга писалась, но без которых она не могла быть написанной. Отдельное спасибо О. Игнатьеву, М. Тихомирову, А. Шапичеву, которые внимательно прочитали условия задач и помогли с составлением тестов к ним. Спасибо моим коллегам В. Гуровцу, М. Густокашину, Д. Кириенко, В. Матюхину, С. Шедову, чей практический опыт в составлении задач для курса обучения программированию, несомненно, оказался полезным. Особое спасибо главному редактору газеты «Информатика» издательского дома «Первое сентября» С. Островскому, который инициировал написание данного пособия для газеты. Организация проверки задач данного курса в сети Интернет была бы невозможна без А. Чернова и А. Шестимерова. Я благодарна всем преподавателям кафедры информатики: А. Анисимову, Т. Богомоловой, И. Гущину, Т. Смыгиной, В. Усатюку, В. Шухардиной, — в совместной работе с которыми рождалась данная книга. И отдельную благодарность хотелось бы высказать моему соавтору по другим изданиям И. Фалиной, любезно согласившейся прочитать рукопись и сделать весьма полезные замечания.

Введение

Словарь языка Pascal

Алфавит любой программы на языке Pascal состоит из *букв*, *цифр* и специальных символов. Из последних образуют *знаки операций* и *знаки пунктуации*.

Буквы: _, A, B, C, D, E, F, G, H, I, J, k, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z.

Обратите внимание, что символ подчеркивания «_» в языке Pascal, как и во многих других языках программирования, является *буквой*.

Цифры: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0.

Знаки операций: +, -, *, /, <, >, <= (меньше или равно), >= (больше или равно) , =, <> (не равно).

Знаки пунктуации:

{} или (* *)	Скобки комментариев
//	Комментарий до конца строки, используется только в Delphi
[]	Выделение индексов массивов, элементов множеств
' '	Выделение символа или строковой константы
( )	Выделение выражений, списков параметров
:=	Знак оператора присваивания
;	Разделение операторов и объявлений
:	Отделение переменной и константы от типа и метки от оператора
=	Отделение имени типа от описания типа или константы от ее значения
,	Запятая для разделения элементов в списке
..	Разделение границ диапазона
.	Конец программы, отделение целой части от дробной, отделение полей в записи
^	Значение величины по ее указателю
@	Обозначение адреса переменной
#	Обозначение символа по его коду
$	Обозначение директивы компилятора или знак 16-ричного числа

Переменные

Любой алгоритм или программа для компьютера состоит из двух разделов: описания *данных* и описания *действий*, которые над этими данными необходимо выполнить. В языках программирования действия представляются так называемыми *операторами*. Данные есть общее понятие для всего того, с чем оперирует компьютер. Память компьютера с точки зрения языка Pascal разделена на секции, называемые *переменными*. Переменные бывают разных *типов*. Тип определяет размер памяти, отводимой под переменную, а также допустимое конечное множество значений, которые может принимать та или иная переменная. Каждая переменная имеет жестко закрепленное за ней имя. Значения переменных могут меняться в ходе выполнения программы.

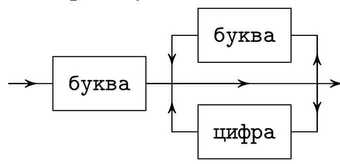
Для обозначения имен переменных, констант, типов, процедур, функций и самой программы используются *идентификаторы*.

Идентификатор — это любое количество *букв* и *цифр*, начинающееся с *буквы*. Идентификатор не может совпадать со служебным словом. Имена могут быть очень длинными, но в конкретных реализациях в них анализируется лишь несколько первых символов (например, 63, то есть имена, у которых первые 63 символа совпадают, считаются одинаковыми). Прописные и строчные буквы в именах не различимы. Так, имена `my_name`, `My_Name` и `mY_nAmE` совпадают.

Список служебных (или зарезервированных, или ключевых) слов (на примере Delphi):

```
and            exept            label            set
array          exports          library          shl
as             file             mod              shr
asm            finalization     nil              string
begin          finally          not              then
case           for              object           threadvar
class          function         of               to
const          goto             or               try
constructor    if               packed           type
destructor     implementation   procedure        unit
dispinterface  in               program          until
div            inherited        property         uses
do             initialization   raise            var
downto         inline           record           while
else           interface        repeat           with
end            is               resourcestring   xor
```

Для описания термина *идентификатор* очень удобно использовать синтаксическую диаграмму:



Синтаксическая диаграмма — это графическое описание синтаксиса языка программирования. С ее помощью можно определить корректность тех или иных конструкций (предложений) с точки зрения конкретного языка программирования. В отличие от блок-схем, у синтаксической диаграммы могут существовать безусловные разветвления, что означает при проверке синтаксиса возможность пойти по любой из ветвей. Могут присутствовать и бесконечные циклы. Так, из приведенной диаграммы следует, что букв и цифр в одном имени может быть как угодно много, а после первой буквы может находиться буква, цифра или ничего больше не стоять (конец диаграммы). В дальнейшем буквы и цифры могут также чередоваться произвольным образом.

Общий вид программы

программа

program имя ; uses используемые модули блок .

блок

begin оператор ; end

label - описание меток

const - описание констант

type - описание типов

var - описание переменных

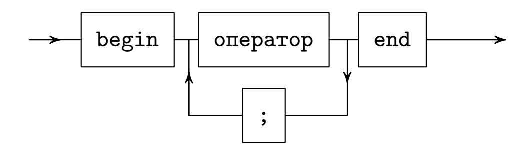
описание подпрограмм

Из диаграмм видно, что программа может иметь, а может не иметь своего имени (заголовка программы). Любой из разделов описаний также может отсутствовать. Перед собственно программой, которая начинается со слова begin, идут все необходимые описания. Располагаться они могут в различном порядке (убедитесь, что диаграмма позволяет возвращаться к разделам описаний, размещенных выше текущего раздела). Точка с запятой служит разделителем между операторами, а не является окончанием каждого оператора. Поэтому перед end точку с запятой можно не ставить.

Приведем синтаксическую диаграмму понятия *оператор*:



Последний из перечисленных операторов применяется в случае, когда на месте *оператора* должны стоять сразу несколько операторов. Приведем диаграмму *составного оператора*:



Слова begin и end в данном случае называют *операторными скобками*.

Урок 1

Простейшая программа на языке Pascal

Собственно программа на языке Pascal (как еще говорят, *тело программы*) должна начинаться со слова `begin`, а заканчиваться словом `end` и знаком точки (см. синтаксическую диаграмму), то есть ее можно сравнить с законченным предложением. Программа, состоящая только из этих слов, разделенных пробелом или переводом строки, верна, но ничего не делает. Добавим в нее вызов процедуры печати каких-либо сообщений (в дальнейшем мы будем называть его *оператором вывода*), например:

```
begin
  write('Hello!')
end.
```

Текст, заключенный в апострофы, компьютер не анализирует, а просто выводит на экран, поэтому он может быть произвольным, то есть содержать любые символы, набираемые на клавиатуре, в том числе русские и латинские большие и маленькие буквы. Программа может быть записана и в одну строку, тогда различные слова нужно разделять хотя бы одним пробелом:

```
begin  write('Hello!') end.
```

Наберите текст этой программы и запустите ее на выполнение. Научитесь просматривать результат работы программы. Иногда для этого удобно вставлять в конец программы оператор `readln`.

Теперь познакомимся с правилами вывода данных в языке Pascal подробнее.

Вывод данных

Стандартные процедуры `write` и `writeln` служат для вывода информации. Правило их использования одно и то же: после слова `write` или `writeln` в скобках через запятую перечисляются параметры, которые мы хотим напечатать. Число этих параметров не ограничено. Запятая служит разделителем между параметрами:

```
writeln(параметр, параметр, ..., параметр)
```

Существует три вида параметров: *константы*, *переменные* и *выражения* (например, арифметические выражения). Константы быва-

ют числовые (это просто различные числа — целые и вещественные), логические и строковые. Любой текст, набранный с клавиатуры и заключенный в апострофы (одиночные кавычки), называется *строковой константой*. Если в текст нам нужно поместить апостроф, например в слове O'key, на этом месте нужно набить два апострофа подряд вместо одного: `write('O''key')`. Все параметры в `write` или `writeln` независимы друг от друга, поэтому в одном и том же операторе могут встречаться параметры разных типов в произвольном порядке.

При выполнении оператора вывода все параметры будут напечатаны в одной строке в том же порядке, в каком они перечислены в списке параметров. Любая константа, числовая или строковая, будет напечатана так, как вы ее написали в вызове `write` или `writeln` (в строковой константе начальный и конечный апострофы напечатаны не будут, а вместо двух апострофов, расположенных в строковой константе подряд, на экране появится в этом месте один); вместо переменной на экране появится ее значение, а вместо арифметического выражения — результат его вычисления.

Между `write` или `writeln` существует единственное различие: после (!!!) выполнения `writeln` курсор переходит на новую строку, а после выполнения `write` курсор остается в той же строке, и новая печать данных с помощью `write` или `writeln` или ввод данных при помощи `read` или `readln` (операторов чтения данных) будет проходить в той же строке.

При печати параметров пробелы между ними автоматически не вставляются, например, при печати чисел 1, 2, 3 с помощью `writeln(1, 2, 3)` все они сольются в одно число — 123. Чтобы разделить выводимые элементы, можно поместить между ними символ пробел, например `writeln(1, ' ', 2, ' ', 3)`, или отформатировать печать, поставив после каждого элемента списка вывода двоеточие и целое число, которое указывает, сколько позиций на экране должна занимать выводимая величина, например `writeln(1:3, 2:3, 3:3)`. Отметим, что элемент дополняется начальными пробелами слева, с тем чтобы соответствовать указанной после двоеточия величине. Результаты выполнения двух последних операторов будут выглядеть так:

```
1 2 3
  1  2  3
```

Если указанное в формате печати число меньше, чем необходимо, то Pascal при выводе увеличит это значение до минимально необходимого размера. При выдаче на экран значений вещественных выра-

жений в формате печати полезно указывать еще один параметр после второго двоеточия. Он будет обозначать количество символов после десятичной точки, которые мы хотим напечатать. Например, при печати результата стандартной функции `pi`, которая с машинной точностью выдает значение числа π, оператор `write(pi:0:0, pi:5:2, pi/2:2:0)` выдаст на экран:

```
3 3.14 2
```

Заметим, что при печати фиксированного количества цифр вещественного числа оно предварительно округляется по правилам математики.

Примеры операторов вывода:

```
write('Нажмите любую клавишу');
writeln(2, '+', 2, '=', 4);
write('7+5', '='); writeln(7 + 5);
```

Задания

1. Определите, что будет выведено на экран после выполнения трех операторов, приведенных в конце урока 1. Напишите соответствующую программу и сверьте результат ее выполнения со своим предположением.

2. Напишите программу, которая семью различными способами будет выдавать на экран фразу «2+2=4» (без кавычек). Воспользуйтесь для этого операторами `writeln`, которые следует разделять друг от друга знаком точка с запятой (;) — разделителем между операторами в языке Pascal.

Первый способ должен содержать всего один параметр в операторе `writeln`, второй — два и т. д., а пятый, шестой и седьмой — пять. При этом шестой оператор не должен содержать числа 4, а седьмой — числа 2. В операторах печати должны использоваться все три вида параметров.

3. Напишите программу, которая выводит на весь экран ваше имя. Линии букв можно составлять из символов «*» или других символов.

Урок 2

Целые и вещественные числовые типы данных

Каждая переменная в программе должна быть *описана*, то есть упомянута в разделе описаний переменных `var` с указанием своего типа. Основным типом для работы с целочисленными данными является тип `integer`. Значениями переменных этого типа являются целые числа от −32768 до 32767 в Borland Pascal и от −2147483648 до 2147483647 в Delphi (в Borland Pascal значения из такого диапазона принимают переменные типа `longint`). К переменным целочисленных типов применимы следующие арифметические операции:

+, −, ∗ — сложение, вычитание и умножение;

`div` — целая часть от деления (значение не округляется, а дробная часть просто отбрасывается, в том числе и для отрицательных чисел);

`mod` — остаток от деления нацело: `a mod b = a - ((a div b) * b)`.

Приведем примеры выполнения двух последних операций для всех возможных знаков аргументов:

```
 5 div  3 =  1;  5 mod  3 =  2;
-5 div  3 = -1; -5 mod  3 = -2;
 5 div -3 = -1;  5 mod -3 =  2;
-5 div -3 =  1; -5 mod -3 = -2;
```

Основным типом для работы с вещественными (действительными) числами является тип `real`. Вещественных чисел, точно представимых в компьютере, конечное число. Остальные числа либо приближаются представимыми, либо оказываются непредставимыми. Последнее относится к слишком большим и к слишком маленьким вещественным числам[1].

К числовым типам данным применимы стандартные функции, представленные в таблице.

В программировании существует негласное правило, что имена целочисленных переменных начинаются с букв i, j, k, l, m, n, а вещественных — с остальных букв. Это правило не применяется, если

[1] Подробнее о представлении чисел в компьютере можно прочитать в книге Е. Андреевой, И. Фалиной «Системы счисления и компьютерная арифметика» или в книге тех же авторов «Математические основы информатики».

Функция	Комментарий	Тип аргумента	Тип результата
`abs(x)`	$\lvert x\rvert$ — модуль числа x	`integer`, `real`	соответствующий
`sqr(x)`	x^2	`integer`, `real`	соответствующий
`sqrt(x)`	$\sqrt{x}$ — корень квадратный из x	`integer`, `real`	`real`
`pi`	`3.1415926535897932385`	нет	`real`
`sin(x)`	$\sin x$	`integer`, `real`	`real`
`cos(x)`	$\cos x$	`integer`, `real`	`real`
`arctan(x)`	$\operatorname{arctg} x$	`integer`, `real`	`real`
`exp(x)`	e^x — экспонента	`integer`, `real`	`real`
`ln(x)`	$\ln x$ — натуральный логарифм числа x	`integer`, `real`	`real`
`round(x)`	округляет x до ближайшего целого	`real`	`integer`
`trunc(x)`	отбрасывает дробную часть числа x	`real`	`integer`
`int(x)`	целая часть аргумента	`real`	`real`
`frac(x)`	дробная часть аргумента	`real`	`real`

переменная имеет мнемоничное имя, то есть ее название отражает смысл хранимой величины.

Считывание значений переменных с клавиатуры

Процедуры `read` и `readln` предназначены для задания значений переменным путем ввода их с клавиатуры или из файла. Правило их применения одно и то же: после слова `read` или `readln` в скобках через запятую перечисляются имена переменных, значения которых мы хотим ввести. Число этих имен не ограничено. Запятая служит разделителем между идентификаторами:

```
readln(имя, имя, ..., имя)
```

При вызове процедуры read или readln выполнение программы будет приостановлено, до тех пор пока пользователь не введет соответствующее количество значений. Вводимые значения должны быть того же типа, что и переменные. Если в read или readln переменных несколько, то они могут быть набиты в одной строке, но одно число от другого должно отделяться пробелом или переводом строки (при вводе символьных переменных это не так). Чтобы ввести набитые значения и выполнить оператор read или readln, нужно нажать клавишу «Enter». В результате переменные приобретут заданные вами значения. Между read и readln существует единственное различие: после выполнения readln курсор переходит на новую строку, а после выполнения read курсор остается в той же строке, и новая набивка данных для read или readln будет проходить в той же строке. Но так как после нажатия клавиши «Enter» курсор в любом случае переходит на новую строчку, для однократного ввода значений переменных разницу между операторами read и readln заметить невозможно. Тем не менее в данном случае лучше использовать readln. Оператор readln можно использовать и без параметров вообще. Тогда программа просто будет находиться в режиме ожидания, пока пользователь не нажмет клавишу «Enter». Такой оператор, например, удобно ставить самым последним оператором в программе. Тогда можно сразу посмотреть результат работы программы, а потом нажать «Enter», и только после этого работа программы завершится.

Замечание. Перед вводом данных с клавиатуры рекомендуется выдавать на экран приглашение, например:

```
write('Введите число a =>'); readln(a);
```

Задания

1. Вычислите:

```
20 div 6        trunc(6.9)
20 mod 6        round(6.9)
20 div 4        trunc(-1.8)
20 mod 4        round(-1.8)
2 div 5         round(0.5)
2 mod 5         round(-0.5)
```

2. Определите тип выражения (integer или real):

```
1+0.0           sqrt(16)
20/4            sin(0)
sqr(4)          trunc(pi)
sqr(5.0)        int(pi)
```

3. Напишите программу, в которой описана одна переменная — a. Программа выдает запрос:

```
Введите значение b =>
```

Далее вводится некоторое значение и программа выдает

```
b=...
```

Вместо многоточия должно стоять введенное значение.

4. Запишите по правилам языка Pascal следующие арифметические выражения:

1) $\dfrac{a}{b\dfrac{c}{d\dfrac{e}{f\dfrac{h}{k}}}}$;

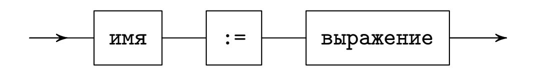
2) $0{,}3\left(\left(\dfrac{\sin^2 x - \cos^2 x}{\sin\dfrac{x+y}{2}} - e^{|\cos x + \sin x|}\right)\ln x - (x-1)^{0{,}5}\right)$.

5. Напишите программу, которая будет вычислять значения 2 в степени 16, 18 и 27 за минимально возможное число умножений.

Урок 3

Оператор присваивания

Синтаксическая диаграмма *оператора присваивания*:



Данный оператор указывает, что нужно вычислить значение выражения и присвоить полученное значение переменной (поместить его в соответствующую секцию памяти) или, как будет показано дальше, имени функции. Типы переменной и выражения должны совпадать. Исключение: переменной любого вещественного типа, например, `real` можно присвоить выражение любого целого типа.

Пример 1. *Приведем пример программы, использующей переменные различных типов*:

```
var a, b, c: real;
    i, j, k: integer;
begin
  a := 1;
  b := 2.0;
  write('i=');
  readln(i);
  j := sqr(i);
  i := i + 1;
  c := sin(a + b);
  writeln('a=', a, 'b=', b, 'c=', c, 'i=', i, 'j=', j)
end.
```

Пример 2. *Рассмотрим фрагмент программы, который демонстрирует способ выделения цифр из произвольного трехзначного числа с использованием целочисленного деления с остатком; трехзначное число находится в целочисленной переменной* `a`:

```
a1 := a div 100; {старшая цифра}
a2 := a div 10 mod 10; {средняя цифра}
a3 := a mod 10; {младшая цифра}
```

Решение следующих задач можно проверять с помощью автоматизированной системы проверки. Но для их решения можно использовать только операторы присваивания, ввода и вывода (соответственно условный оператор или операторы цикла использовать нельзя).

Задачи

1. Напишите программу, которая по введенному не более чем четырехзначному числу k будет выдавать сумму его цифр.

На вход программе подается целое число k ($0 \leqslant k \leqslant 9999$). Выдайте сумму его цифр.

Пример входных данных	Пример выходных данных
2008	10

2. Идет k-я секунда суток. Определите, сколько целых часов h и целых минут m прошло с начала суток. Например, если

$$k = 13257 = 3 \cdot 3600 + 40 \cdot 60 + 57,$$

то $h = 3$ и $m = 40$.

На вход программе подается целое число k ($0 \leqslant k \leqslant 86\,399$). Выведите на экран фразу:

```
It is ... hours ... minutes.
```

Вместо многоточий программа должна выводить значения h и m, отделяя их от слов ровно одним пробелом.

Пример входных данных	Пример выходных данных
13257	It is 3 hours 40 minutes.

3. Часовая стрелка повернулась с начала суток на d градусов. Определите, сколько сейчас целых часов h и целых минут m.

На вход программе подается целое число d ($0 \leqslant d < 360$). Выведите на экран фразу:

```
It is ... hours ... minutes.
```

Вместо многоточий программа должна выводить значения h и m, отделяя их от слов ровно одним пробелом.

Пример входных данных	Пример выходных данных
90	It is 3 hours 0 minutes.

4. Определите, является ли не более чем четырехзначное число k симметричным (например, числа 1331 или 550 являются симметричными, для последнего из них считается, что это четырехзначное число с ведущим нулем).

На вход программе подается целое число k ($0 \leqslant k \leqslant 9999$). Выдайте 1 при положительном ответе на вопрос задачи и любое другое целое число — в противном случае.

Примеры входных данных	Примеры выходных данных
2008	7
2002	1

5. Пусть в школе пять дней в неделю ежедневно проходят шесть уроков. Тогда в неделе всего 30 уроков. По введенному номеру дня d и номеру урока l найдите порядковый номер этого урока в неделе.

На вход программе подаются номер дня d (от 1 до 5) и номер урока l (от 1 до 6). Выведите номер этого урока в неделе (от 1 до 30).

Пример входных данных	Пример выходных данных
2 1	7

6. В книге на одной странице помещается k строк. Таким образом, на 1-й странице печатаются строки с 1-й по k-ю, на второй — с $(k+1)$-й по $(2k)$-ю и т. д. Напишите программу, по номеру строки в тексте определяющую номер страницы, на которой будет напечатана эта строка, и порядковый номер этой строки на странице.

На вход программе подаются число k — количество строк на странице и число n — номер строки в тексте ($1 \leqslant k \leqslant 200$, $1 \leqslant n \leqslant 20\,000$). Выведите два числа — номер страницы, на которой будет напечатана эта строка, и номер строки на этой странице.

Примеры входных данных	Примеры выходных данных
50 1	1 1
20 25	2 5
15 43	3 13

7. Обозначим дни недели числами от 1 (понедельник) до 7 (воскресенье) соответственно. По известному m — дню недели первого числа текущего месяца — определите день недели числа n.

На вход программе подаются 2 целых числа $1 \leqslant n \leqslant 31$, $1 \leqslant m \leqslant 7$. Выведите день недели числа n.

Примеры входных данных	Примеры выходных данных
8 1	1
7 7	6

8. Единица товара стоит a рублей b копеек. Было куплено n штук этого товара. Сколько рублей и копеек пришлось заплатить за всю покупку?

На вход программе подаются три целых числа: $0 \leqslant a \leqslant 30\,000$, $0 \leqslant b < 100$ и $0 \leqslant n \leqslant 30\,000$. Выведите два искомых числа.

Примеры входных данных	Примеры выходных данных
10 15 2	20 30
2 50 4	10 0

9. Цена товара обозначена в рублях с точностью до копеек, то есть вещественным числом с двумя цифрами после десятичной точки, например 10.35. В целочисленных переменных получите и выдайте значения целого числа рублей и целого числа копеек в цене товара.

Пример входных данных	Пример выходных данных
10.35	10 35

10. Даны значения двух моментов времени, принадлежащих одним и тем же суткам: часы, потом минуты и секунды для каждого из моментов времени. Известно, что второй момент времени наступил не раньше первого. Определите, сколько секунд прошло между двумя моментами времени.

В первой строке входных данных находятся три целых числа — часы, минуты и секунды первого момента времени. Во второй строке — три числа, характеризующие второй момент времени. Число часов лежит в диапазоне от 0 до 23, число минут и секунд — от 0 до 59. Выведите число секунд между двумя моментами времени.

Примеры входных данных	Примеры выходных данных
1 1 1 2 2 2	3661
1 2 30 1 3 20	50

Задачи повышенной сложности

1. На вход программе подаются два целых числа m и n, по модулю не превосходящие 10^6. Если m делится на n или n делится на m, то требуется вывести 1, в противном случае — любое другое число.

Примеры входных данных	Примеры выходных данных
2 8	1
0 0	100
5 0	1

2. На вход программе подаются два целых числа m, n, по модулю не превосходящие 10^6. Если $m \geqslant n$, то требуется вывести 1, в противном случае — любое другое число.

Примеры входных данных	Примеры выходных данных
2 8	0
2 1	1

3. Определите, верно ли, что в заданном четырехзначном числе *ровно* две одинаковые цифры.

На вход программе подается целое число k ($1000 \leqslant k \leqslant 9999$). Выдайте 1 при положительном ответе на вопрос задачи и любое другое целое число — в противном случае.

Примеры входных данных	Примеры выходных данных
2008	1
2002	2

4. На вход программе подаются 4 целых числа, по модулю не превосходящие 10^6: m, n, k, l. Если остаток от деления m на n равен k или l, то выведите 1, в противном случае — любое другое число.

Примеры входных данных	Примеры выходных данных
12 8 3 4	1
0 5 1 2	0

5. На вход программе подаются два целых числа n, m, $0 < n \leqslant 12$, $0 \leqslant m < 60$, указывающие момент времени «n часов m минут». Определите наименьшее число полных минут, которое должно пройти до

того момента, когда часовая и минутная стрелки на циферблате совпадут, не обязательно на каком-то делении. Вещественную арифметику не использовать.

Примеры входных данных	Примеры выходных данных
`2 50`	`26`
`3 0`	`16`

6. На вход программе подаются два целых числа n, m, $0 < n \leqslant 12$, $0 \leqslant m < 60$, указывающие момент времени «n часов m минут». Определите наименьшее число полных минут, через которое часовая и минутная стрелки на циферблате расположатся перпендикулярно друг другу. Вещественную арифметику не использовать.

Примеры входных данных	Примеры выходных данных
`2 50`	`10`
`12 0`	`16`

7. На вход программе подаются два числа (не обязательно целые, но не более чем с двумя знаками после десятичной точки). Распечатайте их в порядке возрастания. Используйте только арифметические операции и, при необходимости, стандартные функции.

Примеры входных данных	Примеры выходных данных
`10 35`	`10.00 35.00`
`3.14 2.71`	`2.71 3.14`

8. На вход программе подается вещественное число x. Получите и выведите целое значение функции $\mathrm{sign}(x)$ — знак числа x.

Примеры входных данных	Примеры выходных данных
`3.14`	`1`
`0`	`0`
`-0.5`	`-1`

9. Квадратная таблица размера $n \times n$ заполнена по спирали по часовой стрелке идущими подряд натуральными числами начиная с левой верхней угловой клетки $(1, 1)$. По числу, записанному в некоторой клетке, определите значения строки и столбца этой клетки (они нумеруются, начиная с 1, сверху и слева соответственно).

На вход программе подаются значение n ($1 \leqslant n \leqslant 30\,000$) и значение, стоящее в искомой клетке (от 1 до n^2). Выведите значения строки и столбца этой клетки.

Примеры входных данных	Примеры выходных данных
`5 1`	`1 1`
`100 2138`	`95 36`

10. Квадратная таблица размера $n \times n$ заполнена по спирали по часовой стрелке идущими подряд натуральными числами, начиная с левой верхней угловой клетки (1, 1). По номерам строки и столбца некоторой клетки (они нумеруются, начиная с 1, сверху и слева соответственно) определите, какое число в ней записано.

На вход программе подаются значение n ($1 \leqslant n \leqslant 30\,000$) и значения строки и столбца (от 1 до n). Выведите число, записанное в этой клетке.

Примеры входных данных	Примеры выходных данных
`5 1 1`	`1`
`100 95 36`	`2138`

Урок 4

Логический тип данных. Условный оператор

Множество значений логического типа `boolean` содержит всего два элемента — `false` (ложь) и `true` (истина). Эти константы предопределены так, что `false < true`. Логические значения получаются также в результате выполнения операций сравнения числовых, символьных, строковых или логических переменных: =, <>, <, >, <=, >=. Такие сравнения представляют собой частный случай *логических выражений* — выражений со значениями типа `boolean`. Подобные выражения можно присваивать переменным типа `boolean`, а также печатать (на экран будет выведено слово `false` или `true` соответственно). Кроме операций сравнения для построения логических выражений используются операции `not`, `and`, `or`, `xor`. Последняя операция при применении ее к логическим операндам совпадает с операцией «не равно», то есть `(x xor y) = (x <> y)`. Приведем таблицы результатов этой и других логических операций для всех возможных значений операндов (в алгебре логики такие таблицы называются таблицами истинности):

x	not x
false	true
true	false

x	y	x and y	x or y	x xor y
false	false	false	false	false
false	true	false	true	true
true	false	false	true	true
true	true	true	true	false

Логический результат дает также стандартная функция `odd(x)`, которая применяется к целочисленному аргументу *x*:

`odd(x) = true`, если *x* нечетно;

`odd(x) = false`, если *x* четно.

Логические выражения

В логических (булевских) выражениях могут встречаться как арифметические операции, так и логические. Порядок выполнения опера-

ций определяется их приоритетом (в других языках программирования приоритеты операций могут быть другими):

1) `not`;
2) `*`, `/`, `div`, `mod`, `and`;
3) `+`, `-`, `or`, `xor`;
4) `=`, `<>`, `<`, `>`, `<=`, `>=`.

Операции с одинаковым приоритетом выполняются по порядку слева направо. Для изменения порядка выполнения операций применяют круглые скобки.

При вычислении булевских выражений их значение может стать известным еще до конца вычисления всего выражения. Например:

```
x := 0;
b := (x > 0) and (x < 10)
```

Уже после вычисления первого операнда операции `and` ясно, что результат всего выражения — `false`, поэтому второй операнд вычисляться не будет.

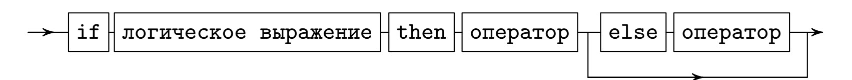
Нередко при составлении программ со сложными логическими выражениями нужно строить их отрицания. Для этого полезно воспользоваться следующей таблицей и тождествами, известными из алгебры логики:

Условие	Противоположное условие
`a < b`	`a >= b`
`a > b`	`a <= b`
`a = b`	`a <> b`

```
not not a = a;
not(a and b) = not a or not b;
not(a or b) = not a and not b.
```

Условный оператор

Синтаксическая диаграмма *условного оператора*:



На месте *оператора* может стоять любой из операторов, в том числе и условный, и составной, но оператор должен быть только один.

Полный условный оператор выполняется так: сначала проверяется условие (вычисляется значение логического выражения), если оно

истинно (равно true), то компьютер выполняет оператор, стоящий после then, если же ложно (равно false), то есть справедливо противоположное условие, то компьютер выполняет оператор, стоящий после else.

Пример 1. *Покажем, как с помощью условного оператора определить четность числа k*:

```
if k mod 2 = 0 then writeln(k, 'четное')
               else writeln(k, 'нечетное')
```

Условный оператор может быть и неполным (укороченным), то есть не содержать слово else и следующий за ним оператор. Тогда, если условие, стоящее после if, ложно, то управление передается оператору, следующему за данным условным.

Пример использования укороченного условного оператора:

```
x := 0;
if x > 0 then writeln('x = ', x)
```

В этом случае ничего напечатано не будет.

Согласно синтаксису условного оператора (см. синтаксическую диаграмму) как после then, так и после else может стоять только один оператор, поэтому при необходимости использовать несколько операторов используется составной оператор. Например:

```
if x < 0 then
begin
  i := i + 1;
  x := x - 1
end
else i := i - 1
```

Здесь для случая $x < 0$ будут выполнены два оператора, а для противоположного случая ($x \geqslant 0$) — один оператор.

Если после then в качестве оператора стоит условный оператор, то возможна такая конструкция:

```
if условие1 then
  if условие2 then оператор1
  else оператор2
```

В этом случае непонятно, к какому if относится else, то есть какой из условных операторов полный, а какой укороченный. Для таких случаев введено правило, по которому else всегда относится к ближайшему if. Значит, в приведенной конструкции первый оператор укороченный (без else), а второй — полный. Если же с точки зрения логики программы требуется, чтобы первый оператор был полным,

а второй — укороченным, то следует использовать операторные скобки:

```
if условие1 then
begin
  if условие2 then оператор1
end
else
  оператор2
```

Благодаря операторным скобкам `else` теперь относится к первому `if`, а не ко второму. Общий способ решения этой проблемы — всегда использовать только полные условные операторы:

```
if условие1 then
  if условие2 then оператор1
  else {здесь стоит пустой оператор}
else оператор2
```

Такая конструкция всегда будет однозначной.

Пример 2. *Рассмотрим примеры вложенных условных операторов*:

```
1) if i = 1 then
     if j = 1 then writeln('i = j = 1')
     else writeln('i = 1, j <> 1')
2) if i = 1 then
     if j = 1 then writeln('i = j = 1')
     else writeln('i = 1, j <> 1')
   else writeln('i <> 1')
```

В обоих случаях будет напечатано то условие, при котором мы попадаем на данный оператор печати. При любых значениях i и j для каждой из программ выполнится только один из `writeln`.

Приведем несколько замечаний, полезных при использовании условного оператора:

1. Перед `else` знак ; не ставится никогда!!!

2. Рассмотрим следующий фрагмент программы:

```
if b then ;
begin
  s1;
  s2
end
```

В данном случае составной оператор будет выполняться всегда, так как после `then` стоит пустой оператор.

3. Пусть n условий исчерпывают все возможные случаи, например $x<2$, $x=2$, $x>2$. Тогда вызов операторов, соответствующих каждому из случаев, можно запрограммировать двумя способами:

```
if x < 2 then s1;   | if x < 2 then s1
if x = 2 then s2;   | else if x = 2 then s2
if x > 2 then s3    |              else s3
```

Во втором способе не делаются лишние проверки, но первый способ нагляднее. Постарайтесь понять, почему во втором случае вообще не проверяется условие `x > 2`.

4. Рассмотрим следующий условный оператор:

```
if a = c then b := true
         else b := false
```

Такая запись является избыточной. Вместо условного оператора здесь можно использовать логическое выражение:

```
b := a = c
```

Аналогично, вместо

```
if b = true then оператор
```

логичнее писать

```
if b then оператор
```

Задачи

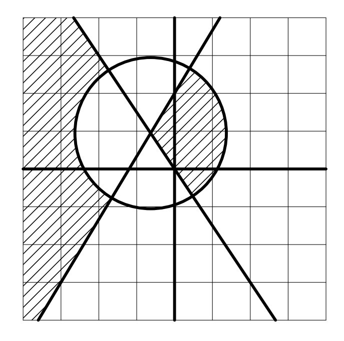
1. На выданном вам рисунке на координатной плоскости изображены окружности, прямые, параболы (некоторые из упомянутых линий могут отсутствовать). Несколько областей, ограниченных этими линиями, заштрихованы. Данные линии можно описать уравнениями в декартовой прямоугольной системе координат (x, y) на плоскости:

$$y = f(x), \quad \text{или } x = f(y), \quad \text{или } f(x, y) = 0.$$

На рисунке изображены также оси координат и координатная сетка, линии которой проведены через единицу масштаба. Восстановите по рисунку уравнения всех линий, изображенных на нем.

Напишите программу, которая позволит определять, принадлежит ли точка с вещественными координатами $(x_0,\ y_0)$ фигуре, состоящей из всех заштрихованных на рисунке областей, или нет. Точки на границах областей не рассматривать, то есть ваша программа может считать их как принадлежащими фигуре, так и не принадлежащими ей. Значения x_0, y_0 вводятся с клавиатуры. В качестве ответа программа выдает на экран `true`, если точка с введенными координатами $(x_0,\ y_0)$ принадлежит заштрихованной фигуре, и `false`, если точка $(x_0,\ y_0)$ не принадлежит ей. Условный оператор не использовать.

Пример рисунка:



Примеры входных данных	Примеры выходных данных
`1 1`	`true`
`1 -1.5`	`false`

2. Напишите программу, которая будет считывать значение вещественной переменной x и выдавать значение следующей функции от x:

$$sign(x) = \begin{cases} -1, & x < 0; \\ 0, & x = 0; \\ 1, & x > 0. \end{cases}$$

Примеры входных данных	Примеры выходных данных
`0.1`	`1`
`-1000000000.5`	`-1`

3. Напишите программу, которая будет считывать значения целых переменных a, b и c и распечатывать их в порядке возрастания. Значения a, b и c по модулю не превосходят 30 000.

Решите задачу двумя способами:

1) не используя операторы присваивания и логическую операцию `and`;

2) используя операторы присваивания, но не используя `and`.

В первом случае нужно в зависимости от значений переменных печатать их в соответствующем порядке, а во втором — значения нужно переместить так, чтобы в переменной a оказалось минимальное зна-

чение, в b — среднее, а в c — максимальное. Оператор печати в этом случае должен быть только один: `writeln(a, b, c)`.

Примеры входных данных	Примеры выходных данных
`1 2 3`	`1 2 3`
`-1 -2 -3`	`-3 -2 -1`

4. Напишите программу для решения уравнения $ax = b$ относительно x в целых числах. Учтите, что a может принимать любые значения, в том числе и 0.

На вход программе подаются целые числа a и b, по модулю не превосходящие 30 000. Требуется вывести целый корень уравнения, если он существует и единственный. Если уравнение не имеет целых корней, то выведите фразу `no solution`. Если уравнение имеет больше одного целого корня, то выведите `many solutions`.

Примеры входных данных	Примеры выходных данных
`1 -2`	`2`
`2 -1`	`no solution`

5. Даны координаты точки на плоскости. Требуется определить, в какой координатной четверти она лежит. Вводятся два целых, не равных нулю числа, по модулю не превосходящие 30 000: координаты точки плоскости (x, y). Выведите номер координатной четверти, в которой лежит эта точка (1, 2, 3 или 4).

Примеры входных данных	Примеры выходных данных
`1 -2`	`4`
`2 1`	`1`

6. Поле шахматной доски определяется парой натуральных чисел, каждое из которых не превосходит 8. По введенным координатам двух полей (k, l) и (m, n) выясните, угрожает ли ферзь, находящийся на поле (k, l), полю (m, n)?

На вход программе подаются 4 целых числа k, l, m и n. Выведите `YES` или `NO` в зависимости от ответа на вопрос задачи.

Примеры входных данных	Примеры выходных данных
`1 1 2 2`	`YES`
`1 1 2 3`	`NO`

7. Поле шахматной доски определяется парой натуральных чисел, каждое из которых не превосходит 8. По введенным координатам двух полей (k, l) и (m, n) выясните, являются ли эти поля полями одного цвета?

На вход программе подаются 4 целых числа k, l, m и n. Выведите `YES` или `NO` в зависимости от ответа на вопрос задачи.

Примеры входных данных	Примеры выходных данных
`1 1 2 2`	`YES`
`1 1 2 3`	`NO`

8. По введенному номеру года — натуральному числу, не превосходящему 10 000, требуется определить, является ли данный год високосным. Напомним, что високосными являются года, номера которых кратны 4, но не кратны 100, а также года, номера которых кратны 400. Выведите `YES` или `NO` в зависимости от ответа на вопрос задачи.

Примеры входных данных	Примеры выходных данных
`2007`	`NO`
`2000`	`YES`

9. Узник замка Иф.

За многие годы заточения узник замка Иф проделал в стене прямоугольное отверстие размером $D \times E$. Замок Иф сложен из кирпичей размером $A \times B \times C$. Узник хочет узнать, сможет ли он выбрасывать кирпичи в море из этого отверстия, для того чтобы сделать подкоп. Помогите ему, считая, что стороны кирпича будут параллельны сторонам отверстия.

На вход программе подаются 5 чисел A, B, C, D, E. Все числа натуральные, не превосходящие 10 000. Выведите `YES` или `NO` в зависимости от ответа на вопрос задачи.

Примеры входных данных	Примеры выходных данных
`1 1 1 1 1`	`YES`
`2 2 2 1 1`	`NO`

10. Даны три натуральных числа — длины стороны треугольника. Определите, существует ли треугольник с такими сторонами, и если он существует, то определите его тип (остроугольный, тупоугольный, прямоугольный).

На вход программе подаются 3 натуральных числа, не превосходящих 10 000. Необходимо вывести одно из слов: `rectangular` для прямоугольного треугольника, `acute` для остроугольного треугольника, `obtuse` для тупоугольного треугольника или `impossible`, если треугольник с указанными сторонами не существует.

Примеры входных данных	Примеры выходных данных
`3 4 5`	`rectangular`
`1 2 3`	`impossible`

11. Решите в действительных числах уравнение $ax^2 + bx + c = 0$.

На вход программе подаются целые числа a, b, c, по модулю не превосходящие 30 000. Выдайте код ситуации и значения корней:

−1 — бесконечное множество решений;

0 — нет действительных корней;

1 — уравнение вырождается в линейное, выдать x;

2 — уравнение квадратное, два различных корня, выдать x_1 и x_2;

3 — уравнение квадратное, кратный корень, выдать x.

Значения корней выводить в порядке возрастания с двумя знаками после десятичной точки.

Примеры входных данных	Примеры выходных данных
`0 0 0`	`-1`
`1 -2 1`	`3 1.00`

Задачи повышенной сложности

1. На столе лежат коробка размера $A_1 \times B_1 \times C_1$ и коробка размера $A_2 \times B_2 \times C_2$. Выясните, можно ли одну из этих коробок положить в другую, если разрешены повороты коробок вокруг любого ребра на угол 90°.

Первая строка входных данных содержит три целых числа A_1, B_1 и C_1. Вторая строка входных данных содержит три целых числа A_2, B_2 и C_2. Все числа натуральные и не превосходят 1000.

Если коробки одинаковы, выведите:

```
Boxes are equal
```

Если первая коробка может быть положена во вторую, выведите:

```
The first box is smaller than the second one
```

Если вторая коробка может быть положена в первую, выведите:

```
The first box is larger than the second one
```

В остальных случаях выведите:

```
Boxes are incomparable
```

Примеры входных данных	Примеры выходных данных
1 2 3 3 2 1	Boxes are equal
3 4 5 2 4 6	Boxes are incomparable

2. Яша плавал в бассейне размером $n \times m$ метров и устал. В этот момент он обнаружил, что находится на расстоянии x метров от одного из длинных бортиков (не обязательно от ближайшего) и y метров от одного из коротких бортиков. Какое минимальное расстояние должен проплыть Яша, чтобы выбраться из бассейна на бортик?[1]

На вход программе подаются 4 натуральных числа n, m, x, y ($n \neq m$), разделенные пробелами. Все числа не превосходят 100. Требуется вывести одно число — минимальное расстояние, которое должен проплыть Яша, чтобы выбраться на бортик.

Пример входных данных	Пример выходных данных
10 25 7 8	3

3. Узник замка Иф-2.

За многие годы заточения узник замка Иф проделал в стене прямоугольное отверстие размером $D \times E$. Замок Иф сложен из кирпичей размером $A \times B \times C$. Узник хочет узнать, сможет ли он выбрасывать кирпичи в море из этого отверстия, для того чтобы сделать подкоп. Помогите ему, считая, что стороны кирпича могут произвольно располагаться относительно сторон отверстия.

На вход программе подаются 5 чисел A, B, C, D, E. Все числа натуральные, не превосходящие 10 000. Выведите `YES` или `NO` в зависимости от ответа на вопрос задачи.

Примеры входных данных	Примеры выходных данных
1 1 1 1 1	YES
2 2 2 1 1	NO

4. По координатам трех точек на плоскости требуется определить их взаимное расположение.

[1] Эта задача предлагалась на московской командной олимпиаде для восьмиклассников в 2006 г., автор В. Гуровиц.

На вход программе подаются 6 чисел: x_1, y_1, x_2, y_2, x_3, y_3. Все числа целые, по модулю не превосходят 100. Они задают 3 точки плоскости: $a(x_1, y_1)$, $b(x_2, y_2)$, $c(x_3, y_3)$. Следует определить взаимное расположение точек и выдать на экран код ситуации:

0 — 3 точки совпадают;

1 — ровно 2 точки из трех совпадают;

2 — точки не совпадают, лежат на одной прямой;

3 — точки образуют остроугольный треугольник;

4 — точки образуют прямоугольный треугольник;

5 — точки образуют тупоугольный треугольник.

Постарайтесь использовать как можно меньше логических операций (операций сравнения и логических связок).

Примеры входных данных	Примеры выходных данных
`1 1 2 2 3 3`	`2`
`1 2 1 2 1 2`	`0`

5. Задана система двух линейных уравнений относительно x и y:

$$\begin{cases} ax+by=e, \\ cx+dy=f. \end{cases}$$

Требуется решить данную систему. На вход программе подаются 6 чисел: a, b, c, d, e, f (все числа целые, по модулю не превосходят 100; обратите внимание на то, что сначала вводятся значения a, b, c, d, а потом — e и f). Выдайте описание решения в следующем виде:

`0` — решений нет;

`1` — решение имеет вид $y=kx+b$, $k\neq 0$;

`1X` c — решение представляет собой пары вида (x, c), c фиксировано, x любое; вывести c с точностью две цифры после десятичной точки;

`1Y` c — решение представляет собой пары вида (c, y), c фиксировано, y любое; вывести c с точностью две цифры после десятичной точки;

`2` x y — решение системы единственно; вывести x и y с точностью две цифры после десятичной точки;

`2XY` — любая пара (x, y) является решением данной системы.

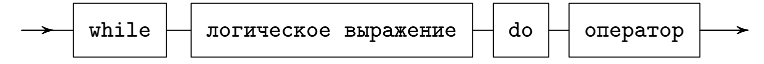
Примеры входных данных	Примеры выходных данных
`1 0 0 1 3 3`	`2 3.00 3.00`
`1 1 2 2 1 2`	`1`
`0 2 0 4 1 2`	`1X 0.50`

Урок 5

Циклы с предусловием и постусловием

Цикл с предусловием

Многократно повторяемые действия могут быть заданы с помощью оператора цикла. Рассмотрим синтаксическую диаграмму одного из таких операторов — *оператора цикла с предусловием*:



Выполнение оператора цикла сводится к повторному выполнению *оператора* (тела цикла), который мы обозначим S, пока значение логического выражения (B) истинно (до тех пор пока оно не станет ложным). Фактически подобные операторы цикла реализуют повторное выполнение условных операторов

```
if B then S;
```

пока истинно условие B. Если выполняемый оператор не изменяет значения переменных, входящих в условие, то условие будет истинным всегда и цикл будет выполняться вечно, при этом говорят, что программа зацикливается. Если же при первой проверке условия оно сразу оказывается ложным, то оператор цикла не выполняется вообще.

Если в цикле нам необходимо выполнять больше чем один оператор, то, как и в случае с условным оператором, применяется составной оператор, то есть несколько операторов заключаются в операторные скобки `begin` … `end`.

Пример оператора цикла с предусловием:

```
while x <= 0 do x := x + 1;
```

Если до оператора цикла значение `x` было положительно, то цикл не будет выполняться вообще. Если `x` было равно 0, то оператор цикла выполнится ровно один раз, а если `x` было меньше 0, то оператор цикла выполнится несколько раз и закончится, когда `x` станет равным 1.

Пример 1. *По заданному целому неотрицательному значению* n*, не применяя формулы, требуется вычислить* $s=1+2+3+4+\ldots+n$*. Приведем фрагмент программы решения этой задачи*:

```
readln(n);
s := 0;
```

```
i := 0;
while x < n do
begin
  i := i + 1;
  s := s + i
end;
```

В таких задачах очень важно правильно задать до цикла значения изменяемых в цикле переменных и проконтролировать, нужное ли количество раз выполнится цикл. Так, если в рассмотренной задаче заменить условие `x < n` на `x <= n`, то на последнем шаге цикла к `s` прибавится значение `n + 1`, что неверно. Заметим, что приведенная в качестве решения задачи программа автоматически работает верно и для случая `n = 0`. Цикл при этом просто не будет выполняться.

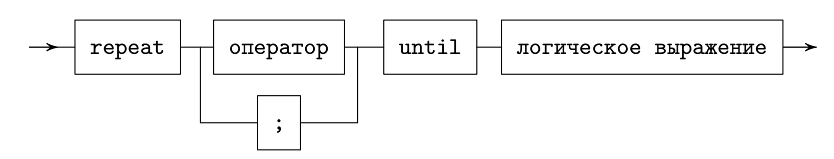
Пример 2. *С клавиатуры вводятся натуральные числа. Последовательность этих чисел заканчивается нулем* (*в данном случае* 0 *не входит в последовательность, это признак окончания последовательности*). *Требуется найти их сумму. Для решения этой задачи можно использовать следующий фрагмент программы*:

```
read(a);
s := 0;
while a <> 0 do
begin
  s := s + a;
  read(a)
end;
```

Для того чтобы эта программа работала, числа надо набивать или в одной строке, разделяя их пробелами, или — каждое в новой строке. Пока не ввести в качестве одного из чисел 0, программа не закончится. Обратите внимание на то, что если первое же число будет нулем, то цикл выполняться не будет.

Цикл с постусловием

В языке Pascal существует еще один оператор цикла с условием, которое проверяется уже после выполнения оператора. Приведем его синтаксическую диаграмму:



В данном операторе слова `repeat` и `until` служат операторными скобками и `begin end` использовать не требуется. На первом шаге цикла операторы, заключенные между `repeat` и `until`, выполняются в любом случае, дальше же цикл будет повторяться, пока значение логического выражения ложно. Цикл закончит свою работу, когда логическое выражение станет истинным. В отличие от цикла с предусловием, здесь истинное значение — это условие окончания цикла.

Пример 3. *Решим задачу из примера* 2 *с помощью цикла с постусловием*:

```
s := 0;
repeat
  read(a);
  s := s + a
until a = 0;
```

Здесь на последнем шаге к значению суммы прибавляется нулевое значение, что ее не меняет.

Задачи

1. По заданным вещественному значению x и целому значению n вычислите x^n (операция возведения в степень в языке Pascal отсутствует). Для решения задачи используйте последовательное домножение результата на x.

На вход программе подаются вещественное число x, по модулю не превосходящее 10, и целое число n, по модулю не превосходящее 20. Выведите значение x^n с точностью три цифры после десятичной точки.

Примеры входных данных	Примеры выходных данных
`2 10`	`1024.000`
`2 -3`	`0.125`

2. На вход программе подаются три целых неотрицательных числа x, n и p, не превосходящих 2×10^9. Кроме того $p > 0$. Требуется вычислить значение x в степени n по модулю p. (операция возведения в степень в языке Pascal отсутствует). Для решения задачи используйте алгоритм эффективного возведения в степень.

Алгоритм основан на тождестве $x^{2n} = x^n x^n$. Тогда если $n = 2^k$, то значение x^n можно получить из x, домножая результат сам на себя k раз (таким образом мы будем последовательно получать значения 2-й, 4-й, 8-й, …, 2^k-й степеней числа x). В свою очередь, произвольное n можно представить как сумму степеней двойки (фактически

перевести n в двоичную систему счисления): $n = 2^{k1} + 2^{k2} + \ldots$ Соответственно $x^n = x^{2^{k1}} \cdot x^{2^{k2}} \cdot \ldots$ Фактически алгоритм быстрого возведения в степень сводится к последовательному получению 2-й, 4-й, 8-й, и т. д. степеней числа x и перемножению необходимых степеней. При этом все операции умножения нужно выполнять по модулю p.

Пример входных данных	Пример выходных данных
`2 10 100`	`24`

3. Найдите сумму цифр введенного целого числа. (На каждом шаге выделяется последняя цифра числа, а затем число делится на 10. Процесс повторяется, пока число не станет равным 0.)

На вход программе подается целое неотрицательное число $n \leqslant 10^9$. Выведите сумму его цифр.

Примеры входных данных	Примеры выходных данных
`1234`	`10`
`5`	`5`

4. На вход программе подается натуральное число $n \leqslant 10^9$. Проверьте, является ли оно простым. Выведите `YES` или `NO` в зависимости от ответа на вопрос задачи. Максимальное время работы программы на одном тесте — 0,1 секунды.

Примеры входных данных	Примеры выходных данных
`13`	`YES`
`10`	`NO`

5. На вход программе подается натуральное число $n \leqslant 10^9$. Проверьте, можно ли представить его в виде суммы двух квадратов натуральных чисел. Выведите `YES` или `NO` в зависимости от ответа на вопрос задачи. В случае положительного ответа во второй строке выведите два числа, сумма квадратов которых равна n. Числа следует выводить в порядке неубывания. Максимальное время работы программы на одном тесте — 0,1 секунды.

Примеры входных данных	Примеры выходных данных
`100`	`YES` `6 8`
`11`	`NO`

6. Вкладчик положил на банковский счет n рублей. Каждый год на сумму вклада начисляется k процентов годовых (будем считать, что процент всегда округляется до целого числа рублей по формуле $[xk/100]$, где x — сумма вклада на начало года). Начисленные проценты добавляются к сумме вклада. Через сколько лет сумма вклада станет не менее m рублей?

На вход программе подаются три натуральных числа: $n \leqslant 10^6$, $k \leqslant 100$, $m \leqslant 1000n$. Выведите одно число — искомое количество лет.

Примеры входных данных	Примеры выходных данных
`100 10 111`	`2`
`100 1 100`	`0`

7. Вкладчик положил на банковский счет n рублей. Каждый год на сумму вклада начисляется k процентов годовых (будем считать, что процент всегда округляется до целого числа рублей по формуле $[xk/100]$, где x — сумма вклада на начало года. Начисленные проценты добавляются к сумме вклада. Через сколько лет сумма вклада как минимум удвоится?

На вход программе подаются два натуральных числа: $n \leqslant 10^6$ и $k \leqslant 100$. Выведите одно число — количество лет, через которое сумма вклада как минимум удвоится.

Пример входных данных	Пример выходных данных
`100 10`	`8`

8. На вход программе подаются два целых неотрицательных числа, одновременно не равных 0, n, $m \leqslant 10^9$. Выведите их наибольший общий делитель. Для решения задачи используйте алгоритм Евклида, основанный на следующем тождестве: НОД(n, m) = НОД(m, r), где r — остаток от деления n на m. Если $r = 0$, то m = НОД(n, m).

Примеры входных данных	Примеры выходных данных
`24 16`	`8`
`11 13`	`1`

9. На вход программе подается последовательность целых чисел, заканчивающаяся числом 0. Выведите минимальное и максимальное значения среди чисел этой последовательности, 0 при этом не учитывается.

При решении задачи массив использовать нельзя.

Примеры входных данных	Примеры выходных данных
`1 -1 0`	`-1 1`
`1 2 3 4 5 0`	`1 5`

10. На вход программе подается последовательность целых чисел, заканчивающаяся числом 0. Выведите их среднее арифметическое с точностью до двух знаков после десятичной точки, 0 при этом членом последовательности не считается.

При решении задачи массив использовать нельзя.

Примеры входных данных	Примеры выходных данных
`1 -1 0`	`0.00`
`1 2 3 4 0`	`2.50`

11. Программа получает на вход последовательность целых чисел, по модулю не превосходящих 10^9. Признак окончания последовательности — число -2×10^9. Программа должна определить вид последовательности — возрастающая, убывающая, случайная или постоянная.

В качестве ответа следует выдать прописными латинскими буквами тип последовательности:

ASCENDING (строго возрастающая);

WEAKLY ASCENDING (нестрого возрастающая, то есть неубывающая);

DESCENDING (строго убывающая);

WEAKLY DESCENDING (нестрого убывающая, то есть невозрастающая);

CONSTANT (постоянная);

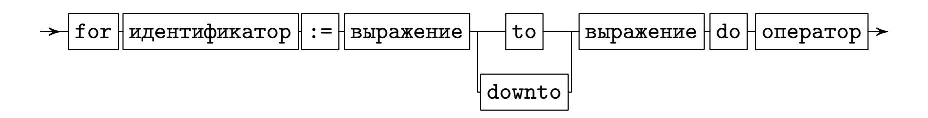
RANDOM (случайная).

При решении задачи массивы использовать нельзя.

Примеры входных данных	Примеры выходных данных
`5 1 -1 -2000000000`	`DESCENDING`
`1 2 2 4 -2000000000`	`WEAKLY ASCENDING`
`1 2 -2 4 -2000000000`	`RANDOM`

Урок 6

Оператор цикла с параметром



Синтаксическую диаграмму для данного оператора необходимо дополнить следующими правилами.

1. Идентификатор и оба выражения должны быть одного и того же порядкового типа. (Из изученных ранее порядковыми являются все целые типы, а также тип `boolean`; далее будут рассмотрены символьный и перечислимый порядковые типы.)

2. Оба выражения вычисляются *до* выполнения оператора цикла и впоследствии не перевычисляются!!!

3. Идентификатор является параметром цикла и по стандарту не должен изменяться внутри оператора цикла (данное требование стандарта поддерживается в языке Delphi). Однако изменение параметра цикла внутри цикла не противоречит синтаксису Borland Pascal, но может приводить к непредсказуемым последствиям, например зацикливанию.

4. После окончания цикла значение параметра цикла не определено, то есть нельзя рассчитывать, что значение параметра равно значению второго выражения.

Оператор цикла выполняется так: сначала вычисляются значения *выражений*; обозначим их A и B. Они являются начальным и конечным значениями параметра цикла соответственно. Если для цикла с `to` $A \leqslant B$, то параметр цикла последовательно будет принимать значения, равные A, $A+1$, $A+2$, …, B, то есть цикл будет выполняться ровно $B-A+1$ раз. Если $A > B$, то цикл не будет выполняться совсем. Если при входе в цикл с `downto` выполняется неравенство $A \geqslant B$, то параметр цикла последовательно будет принимать значения, равные A, $A-1$, $A-2$, …, B, то есть цикл будет выполняться ровно $A-B+1$ раз. Если $A < B$, то цикл не будет выполняться совсем.

Оператор цикла с параметром следует применять, если заранее известно, сколько раз нужно выполнить некоторый оператор. Пара-

метр цикла может являться просто счетчиком, контролирующим количество повторений оператора, а может и использоваться в самом операторе (с учетом того факта, что на каждом шаге цикла параметр цикла на единицу отличается от своего предыдущего значения).

Пример 1. *По заданному целому неотрицательному значению n и вещественному значению x требуется вычислить x^n.*

Решение:

```
p := 1;
for i := 1 to n do p := p * x;
```

Пример 2. *По заданному целому неотрицательному значению n требуется вычислить $n!$. Фрагмент программы отличается от предыдущего фактически в одном (последнем) символе:*

```
f := 1;
for i := 1 to n do f := f * i;
```

Если в качестве оператора цикла необходимо использовать несколько операторов, то применяется составной оператор.

Пример 3. *По заданному натуральному значению n требуется вычислить*

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\ldots\pm\frac{1}{n}.$$

Решение:

```
k := 1;
s := 1;
for i := 2 to n do
begin
  {в переменной k храним знак очередного слагаемого}
  k := -k;
  s := s + k / i
end;
```

Пример 4. *По заданному натуральному значению n и вещественному значению x требуется вычислить $x+x^2+x^3+\ldots+x^n$.*

Решение:

```
s := 0; z := 1;
for i := 1 to n do
begin
  z := z * x; {в переменной z - очередное слагаемое}
  s := s + z
end;
```

Задачи

1. По заданному натуральному значению n вычислите

$$1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\ldots\pm\frac{1}{2n+1}.$$

На вход программе подается натуральное число $n \leqslant 10\,000$. Выведите с точностью 6 значащих цифр после десятичной точки значение указанного выражения.

Примеры входных данных	Примеры выходных данных
1	0.666667
2	0.866667

2. В последовательности $a_1, a_2, \ldots, a_n$ найдите *номер* самого большого числа.

На вход программе сначала подается натуральное число $n \leqslant 10^6$. В следующей строке следуют n целых чисел, по модулю не превосходящих 30 000, — сами члены последовательности. Выведите номер максимального числа. Если таких чисел несколько, то выведите номер последнего из них. Нумерация чисел начинается с единицы.

Массив в программе не использовать.

Примеры входных данных	Примеры выходных данных
3 3 1 2	1
3 1 1 1	3

3. Дана последовательность, состоящая из n чисел. Выясните, сколько раз в ней встречается минимальное число.

На вход программе сначала подается натуральное число $n \leqslant 10^6$. В следующей строке следуют n целых чисел, по модулю не превосходящих 30 000, — сами члены последовательности. Выведите число, которое является ответом на вопрос задачи.

Массив в программе не использовать.

Примеры входных данных	Примеры выходных данных
3 3 1 2	1
4 1 1 2 1	3

4. На вход программе подается натуральное число $n \leqslant 10^6$. Выведите количество делителей числа n, включая 1 и само число n. Максимальное время работы программы на одном тесте — 0,1 секунды.

Примеры входных данных	Примеры выходных данных
13	2
10	4

5. В некоторых видах спорта выступление каждого спортсмена оценивается несколькими судьями, затем из всех оценок удаляются минимальная и максимальная, а из оставшихся берется среднее арифметическое. Если максимальную или минимальную оценку поставили несколько судей, то удаляется только одна из них.

На вход программе сначала подается натуральное число $n \leqslant 100$ — количество судей. В следующей строке следуют n натуральных чисел, не превосходящих 100, — оценки, выставленные судьями одному из спортсменов. Выведите оценку, которая пойдет в зачет данному спортсмену, с точностью с точностью две цифры после десятичной точки. Массив в программе не использовать.

Примеры входных данных	Примеры выходных данных
3 3 1 2	2.00
4 1 1 2 1	1.00

6. По заданному натуральному n и вещественному x вычислите

$$1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!}.$$

На вход программе подаются натуральное число $n \leqslant 100$ и вещественное число x, по модулю не превосходящее 10. Выведите значение указанного выражения с точностью 6 значащих цифр после десятичной точки.

Примеры входных данных	Примеры выходных данных
10 0	1.000000
100 1	2.718282

7. По заданному натуральному n и вещественному x вычислите

$$1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots \pm \frac{x^{2n}}{(2n)!}.$$

На вход программе подаются натуральное число $n \leqslant 100$ и вещественное число x, по модулю не превосходящее 10. Выведите значение указанного выражения с точностью 6 значащих цифр после десятичной точки.

Примеры входных данных	Примеры выходных данных
`10 0`	`1.000000`
`50 3.1416`	`-1.000000`

8. По заданным натуральным числам n и m вычислите

$$\sqrt{m+\sqrt{2m+\ldots+\sqrt{(n-1)m+\sqrt{nm}}}}.$$

На вход программе подаются натуральные числа $n \leqslant 100$ и $m \leqslant 100$. Выведите значение указанного выражения с точностью 6 значащих цифр после десятичной точки.

Пример входных данных	Пример выходных данных
`100 2`	`2.158477`

9. Числа Фибоначчи определяются следующими формулами:

$$f_0 = f_1 = 1; \quad f_n = f_{n-1} + f_{n-2} \quad \text{при } n \geqslant 2.$$

На вход программе подается целое неотрицательное число $n \leqslant 40$. Выведите n-е число Фибоначчи. Массив в программе не использовать.

Пример входных данных	Пример выходных данных
`4`	`5`

10. Дана последовательность, состоящая из n чисел. Найдите в ней два самых маленьких числа.

На вход программе сначала подается натуральное число $n \leqslant 10^6$. Далее следуют n целых чисел, по модулю не превосходящих 30 000, — сами члены последовательности. Выведите минимальное число и второе по величине число (оно может совпадать с минимальным).

Массив в программе не использовать.

Примеры входных данных	Примеры выходных данных
`3 3 1 2`	`1 2`
`4 1 1 2 1`	`1 1`

11. Вычислите по схеме Горнера[1] значение полинома

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

в точке x_0.

На вход программе сначала подается целое неотрицательное число $n \leqslant 20$, затем вещественное число x_0, по модулю не превосходящее 10 — коэффициенты полинома в порядке от a_n до a_0. Далее следуют $n+1$ вещественных чисел, по модулю не превосходящих 10. Выведите значение полинома с точностью три цифры после десятичной точки.

При решении задачи массив не использовать.

Примеры входных данных	Примеры выходных данных
`1 0` `2.5 1`	`1.000`
`2 0.5` `1 1 1`	`1.750`

[1] О схеме Горнера можно прочитать в книге Е. Андреевой и др. «Математические основы информатики».

Урок 7

Вложенные циклы

Рассмотрим задачи, в которых в качестве оператора цикла используется другой цикл.

Пример 1. *Для заданного натурального числа n требуется найти все такие тройки натуральных чисел a, b, c, что $a+b+c=n$.*

Решение:

```
for a := 1 to n - 2 do
  for b := 1 to n - a - 1 do
  begin
    c := n - a - b;
    writeln(a, '+', b, '+', c, '=', n)
  end;
```

Пример показывает, как можно сокращать количество вложенных циклов. Подумайте, почему параметры циклов изменяются именно так, и почему c всегда будет больше нуля.

Пример 2. *Требуется распечатать все трехзначные числа, в которых есть две одинаковые цифры.*

Решение:

```
for j := 1 to 9 do
  for i := 0 to 9 do
    for k := 0 to 9 do
      if (j = i) or (j = k) or (i = k) then
        writeln(j, i, k);
```

Если порядок выводимых чисел не важен, то эту задачу можно решить и более эффективно:

```
for j := 1 to 9 do
begin
  writeln(j, 0, 0, ' ', j, j, 0, ' ', j, 0, j);
  for i := 1 to 9 do
    writeln(i, i, j, ' ', j, i, i, ' ', i, j, i)
end;
```

Пример 3. *Требуется распечатать все трехзначные числа, в которых вторая цифра больше первой, а третья больше второй.*

Решение:

```
for i := 1 to 7 do
  for j := i + 1 to 8 do
    for k := j + 1 to 9 do
      writeln(i, j, k);
```

В приведенном решении удалось избежать использования условных операторов.

Два последних примера показывают, что в задачах по информатике, для того чтобы получить ответ, не всегда нужно оперировать именно теми объектами, о которых идет речь в условии. В этих задачах совершенно излишним было бы перебирать все трехзначные числа, выделять из них цифры и сравнивать их между собой. Более того, формировать из подходящих цифр число, только для того чтобы его распечатать, тоже не нужно: печать нескольких цифр подряд без разделителей приведет к тому, что зрительно эти цифры как раз и образуют нужное число.

Задачи

1. (Модифицированная задача из примера 1.) Для заданного натурального числа n найдите все такие тройки натуральных чисел a, b, c, что $a+b+c=n$ и $a \leqslant b \leqslant c$. Минимизируйте количество условий, которые используются в программе.

На вход программе подается натуральное число $n \leqslant 10^5$. Выведите количество искомых троек.

Примеры входных данных	Примеры выходных данных
3	1
2	0

2. а) Автобусный билет считается счастливым, если в его шестизначном номере сумма первых трех цифр равна сумме трех последних цифр. Подсчитайте и выведите число счастливых билетов с различными номерами (от 000001 до 999999).

б) Билет считается счастливым, если в его n-значном номере сумма первых $[n/2]$ цифр равна сумме $[n/2]$ последних цифр (при нечетном n центральная цифра в «проверке на счастье» не участвует и может быть любой). Подсчитайте число счастливых билетов с различными n-значными номерами (ведущие нули в номерах возможны, но не существует номера, состоящего из одних нулей).

На вход программе подается натуральное число $n \leqslant 15$. Выведите количество n-значных счастливых билетов.

Примеры входных данных	Примеры выходных данных
1	9
2	9

3. Определите количество различных способов выплаты сдачи в размере n рублей купюрами 10 рублей и монетами 5, 2 и 1 рубль. Например, 5 рублей можно выплатить четырьмя различными способами: $5 = 2 + 2 + 1 = 2 + 1 + 1 + 1 = 1 + 1 + 1 + 1 + 1$.

На вход программе подается натуральное число $n < 100$ — размер сдачи, которую необходимо выплатить. Выведите искомое количество способов выплаты.

Примеры входных данных	Примеры выходных данных
2	2
5	4

4. Напишите программу, которая будет разлагать натуральное число $n > 1$ на простые сомножители.

На вход программе подается натуральное $n \leqslant 2 \times 10^9$. Выведите его разложение на простые множители, располагая их в порядке неубывания так, как показано в примерах.

Примеры входных данных	Примеры выходных данных
5	5=5
12	12=2*2*3

5. Дано число n. Найдите число из диапазона от 1 до n с максимальной суммой своих делителей (включая непростые делители, 1 и само число). Если таких чисел несколько, выведите минимальное из них.

На вход программе подается натуральное $n \leqslant 10^6$. Выведите искомое число. Время работы программы на одном тесте — 0,1 секунды.

Примеры входных данных	Примеры выходных данных
5	4
12	12

6. Цифровой корень натурального числа получается следующим образом. Складываются все цифры данного числа. Процесс повторяется, пока в результате не будет получено однозначное число, которое и называется цифровым корнем числа.

На вход программе подается натуральное число $n \leqslant 10^9$. Выведите его цифровой корень.

Примеры входных данных	Примеры выходных данных
`10`	`1`
`888`	`6`

7. Напишите программу для подсчета числа точек с целочисленными координатами, находящихся внутри и на границе круга с центром в начале координат и заданным радиусом r.

На вход программе подается целое неотрицательное значение радиуса $r \leqslant 10^6$. Выведите количество искомых точек в таком круге.

Пример входных данных	Пример выходных данных
`2`	`13`

8. Решите задачу 7, если центр круга находится в произвольной точке плоскости x, y.

В этом случае на вход программе подаются три вещественных числа r, x и y.

Пример входных данных	Пример выходных данных
`1 0.5 0.5`	`4`

9. Экспериментальным путем определите, факториал каких чисел может быть точно вычислен в 32-битном целом типе (`integer` в Delphi и `longint` в Borland Pascal), 64-битном целом типе (`int64` в Delphi и `comp` в Borland Pascal) и типе `extended` (80-битном вещественном типе, для работы с которым в Borland Pascal нужна директива `{$N+}`). Затем напишите программу, которая будет работать следующим образом. Сначала запрашивается число k — количество факториалов, которые надо вычислить ($k \leqslant 100$). Затем k раз считывается значение n ($n \leqslant 100$). Для очередного n сразу вычисляется значение $n!$; если это можно точно сделать хотя бы в одном из указанных типов данных, то выводятся в строке через пробел найденное значение факториала

и число 1, 2 или 3 соответственно в зависимости от типа данных, в котором можно это значение посчитать точно (выводится минимально возможное значение); если в стандартной компьютерной арифметике точно факториал числа n вычислить невозможно, то выводится только 0.

Пример диалога программы:

```
3 {введено значение k}
3 {введено первое значение n}
6 1 {вывод результата для первого факториала}
15 {введено второе значение n}
1307674368000 2 {вывод результата для второго факториала}
100 {введено третье значение n}
0 {вывод результата для третьего факториала}
```

Пример входных данных	Пример выходных данных
3 3 15 100	6 1 1307674368000 2 0

10. По заданному натуральному числу m вычислите

$$\sqrt{m+\sqrt{2m+\ldots+\sqrt{nm+\ldots}}}.$$

На вход программе подается натуральное число $m \leqslant 100$. Выведите значение указанного выражения с точностью 6 значащих цифр после десятичной точки (известно, что это выражение, состоящее из бесконечного числа вложенных корней, для всех указанных значений m конечно).

Пример входных данных	Пример выходных данных
2	2.158477

Урок 8
Порядковые типы данных

Символьный тип данных

Помимо типов для работы с числовыми и логическими переменными в языке Pascal существует тип для обработки переменных символьного типа. Данный тип называется `char` от английского слова *character* — символ. Интересно, что транскрипция этого слова начинается со звука *к*, а не *ч*, так что устоявшееся в русском языке произношение названия символьного типа — «чар», по-видимому, является некорректным и может быть не понято программистами из англоязычных стран.

Значениями символьного типа `char` являются элементы конечного и упорядоченного множества символов, зависящего от текущей кодовой таблицы. В языке Pascal это множество состоит из 256 символов, пронумерованных от 0 до 255. В число этих символов входят все символы, которые вы можете получить на экране с помощью нажатия какой-либо клавиши или комбинации клавиш, а также некоторые другие символы, в том числе и невидимые. Какие именно символы являются константами данного типа, зависит от того, какая кодовая таблица используется в момент *выполнения* (а не написания) программы, то есть одна и та же программа, например, печатающая изображение всех символов кодовой таблицы, на компьютерах с различными текущими кодировками будет иметь различные результаты работы. Обычно первые 128 символов с кодами от 0 до 127 всегда соответствуют так называемым ASCII-символам, а остальные 128 в различных таблицах используются для кодирования букв того

Символ	Действие/значение	Английское название
№ 7	Подача стандартного звукового сигнала	Beep
№ 9	Табуляция (сдвиг курсора на величину табуляции)	Tab
№ 10	Признак конца строки текстового файла	**E**nd **O**f **L**ine (EOLn)
№ 13	Перевод строки	**L**ine **F**eed (LF)
№ 26	Признак конца файла	**E**nd **O**f **F**ile (EOF)

или иного национального алфавита, символов псевдографики и т. п. Кроме того, первые 32 символа считаются *управляющими*, а остальные — *изображаемыми*, то есть имеющими графическое изображение (пробел, имеющий код 32, относится уже к изображаемым символам). Управляющие символы должны восприниматься устройствами вывода и ввода текста как команды (см. таблицу на с. 61).

Компилятор с языка программирования может обрабатывать управляющие символы определенным в нем нестандартным образом. Например, в языке Delphi все первые 33 символа, включая пробел, считаются разделителями при вводе информации, то есть практически любым из них можно отделять, например, числа при вводе из файла или с клавиатуры.

В тексте программы константы символьного типа записываются двумя способами. Наиболее наглядный из них — это заключение любого изображаемого символа в апострофы, например `'*'`, `'F'`, `'1'`. Для того чтобы таким способом представить сам символ апостроф, его записывают внутри апострофов же дважды: `''''` (на клавиатуре для этого надо четыре раза подряд нажать клавишу апострофа). Второй способ позволяет задавать любые символьные константы, в том числе и соответствующие управляющим символам, по их кодам. В этом случае обозначение константы начинается с символа `#`, за которым следует десятичный код (то есть номер от 0 до 255) соответствующего символа, например `#13`, `#65`. Если же мы считываем значение символьной переменной с клавиатуры или из файла, то вводимый символ должен быть набит уже без апострофов. А если считывается последовательность символов (текст), то они набиваются все подряд без разделителей, так как и пробел, и другие разделители числовых констант также являются значимыми символами.

Множество символьных констант является упорядоченным так, что символ `c1` считается меньше символа `c2`, если код первого символа соответственно меньше кода второго. Строчные и прописные буквы являются различными символами, то есть `'a' <> 'A'`. Объясняется это тем, что в любой кодовой таблице каждой из них соответствует свой код. Все строчные и прописные английские буквы, а также символы десятичных цифр упорядочены между собой, то есть

`'0' < '9'`, `'A' < 'Z'` и `'a' < 'z'`, но `'Z' < 'a'`.

Русские буквы одного регистра упорядочены между собой не во всех кодовых таблицах, но чаще всего это так, за исключением буквы Ё.

Символ `'0'` в таблице ASCII имеет код 48, символ `'A'` (латинская) — 65.

Рассмотрим операции, применимые к переменным и константам символьного типа. Во-первых, значения символьных переменных можно считывать, а значения символьных выражений — печатать. Во-вторых, в силу наличия отношения порядка над символьными выражениями определены все операции отношения: <, >, =, <>, <=, >=. Арифметические действия над этим типом не определены.

С символьным типом связаны следующие функции.

1. Функция `chr(i)` выдает символ, имеющий в кодовой таблице номер `i`. Следовательно, параметром этой функции может быть целочисленное выражение, значение которого находится в диапазоне от 0 до 255. Если номер является константой, можно использовать также символ #: `#65` означает то же самое, что и `chr(65)`.

2. Функция `ord(c)`, наоборот, выдает номер символа `c` в кодовой таблице; здесь `c` — или переменная символьного типа, или символьная константа, или функция, результатом выполнения которой является символ (например, `chr`).

3. Функция `succ(c)` выдает символ, следующий в кодовой таблице за символом `c`. Для последнего символа кодовой таблицы эта функция не определена, то есть с точки зрения данной функции символы не считаются расположенными по кругу, они имеют только линейный порядок; эта функция позволяет обрабатывать символы в циклах.

4. Функция `pred(c)` выдает символ, предшествующий в кодовой таблице символу `c`. Для символа с кодом 0 значение этой функции также не определено.

5. Функция `upcase(c)` переводит символы, обозначающие строчные английские буквы, в прописные, остальные символы (в том числе и соответствующие русским буквам) она оставляет неизменными. Например, `upcase('f')` есть `'F'`, а `upcase('*')` есть `'*'`. Обратной функции, то есть функции, переводящей прописные буквы в строчные, в стандартной библиотеке не существует.

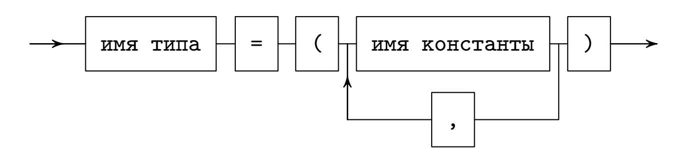
Забегая вперед, отметим, что при анализе констант символьного типа часто удобно использовать так называемые множественные константы, то есть проверку на принадлежность значения символьной переменной некому множеству символов (подробнее см. урок 16). Приведем примеры таких проверок, не вдаваясь в детали синтаксиса:

1)
```
if c in ['a'..'z', 'A'..'Z'] then
    writeln('letter');
```
2)
```
if c in ['1', '3', '5', '7', '9'] then
    writeln('odd digit');
```

Аналогичные конструкции используются и для целочисленных типов.

Перечислимый тип данных

Еще одним скалярным типом данных в языке Pascal является *перечислимый тип*. Данный тип задается программистом путем явного перечисления всех имен, обозначающих значения этого типа. С помощью синтаксической диаграммы порядок описания такого типа в разделе `type` можно определить следующим образом:



В тексте программы все константы данного типа употребляются непосредственно, без апострофов. Все константы пронумерованы согласно описанию начиная с 0, следовательно, определены все операции отношения над переменными и константами данного типа, и можно использовать функцию `ord`. Ввод и вывод для переменных данного типа не предусмотрен.

Уже знакомый вам тип `boolean` фактически является стандартным предопределенным перечислимым типом:

```
type boolean = (false, true);
```

Однако над выражениями данного типа определен расширенный набор операций, а также допустимо их использование в процедурах печати (но считывание не определено).

Перечислимый тип с точки зрения идеологии программирования относится к так называемым флаговым, то есть различные константы одного и того же перечислимого типа используются для обозначения различных ситуаций, каждая из которых должна быть обработана в программе отдельно. Использование таких переменных вместо, например, числовых, делает программу более наглядной и легко отлаживаемой.

Примеры описания порядковых типов:

```
type week = (mon, tues, wed, thur, fri, sat, sun);
     operators = (plus, minus, times, divide);
```

Для того чтобы по номеру константы порядкового типа получить в программе ее значение, можно использовать имя типа:

`boolean(0)` означает `false`;
`week(1)` означает `tues`;
`operators(2)` означает `times`.

Ограниченный тип (диапазон)

Новый тип можно определить, накладывая ограничения на уже определенный ранее (возможно, стандартный) порядковый тип — в этом случае последний называют базовым типом. Ограничение определяется заданием диапазона значений: минимального и максимального значений констант базового типа.



Допустимость операций над значениями ограниченного типа определяется базовым типом. А директива компилятора `{$R+}` позволяет контролировать выход за границу диапазона описанных значений во время выполнения программы.

Именно такие типы данных используются чаще всего для описания индексов у массивов (см. урок 9). Помимо этого они позволяют описывать данные более наглядным образом и контролировать грубые ошибки. Приведем примеры описания ограниченных типов:

```
type age = 0..120;
   digit = 0..9;
type temperature = -50..50;
     age = 0..120;
   digit = 0..9;
  letter = 'a'..'z';
    work = mon..fri;
{в последнем примере используется ограничение на введенный
ранее тип week}
```

Порядковые типы данных

Рассмотрев ранее целые, логические, символьные, перечислимые и ограниченные типы данных мы приходим к их обобщенному понятию — *порядковые типы данных*. Как уже упоминалось ранее, значения любого из таких типов могут быть упорядочены так, что компьютерное представление каждого следующего значения есть двоичное число, на единицу большее предыдущего. Перечислим все стандартные и определяемые пользователем типы, относящиеся к этой группе.

1. Все целые типы: `integer`, `byte`, `shortint`, `smallint` (этот тип в Delphi является 16-битным знаковым типом, что соответствует типу `integer` в Borland Pascal), `word`, `longint` (64-битный целый тип `int64` в Delphi в полном смысле слова порядковым не является).

2. Логический тип `boolean`.

3. Символьный тип `char`.

4. Определяемые программистом перечислимые типы.

5. Ограниченные типы.

Над всеми порядковыми типами определены следующие функции:

1) `ord(a)` — выдает «порядковый номер» `a` среди констант соответствующего типа; для целых типов, в том числе и знаковых, это будет само число, для остальных — номер в нумерации значений типа, начиная с нуля; данная функция по сути позволяет производить преобразование любого порядкового типа в числовой, что иногда очень удобно при решении задач;

2) `succ(a)` — выдает следующее по порядку значение за значением `a` (не определена для максимального значения типа);

3) `pred(a)` — выдает предшествующее значению `a` значение (не определена для минимального значения типа);

4) получение по номеру элемента типа: `<имя типа>(i)`, по-другому это называют приведением к типу; например, `char(65)` соответствует прописной латинской букве A.

Переменные любого порядкового типа можно употреблять в качестве параметров цикла `for`, то есть в цикле можно перебирать значения не только числовых, но и символьных, логических и других параметров порядкового типа. Кроме того, для обработки различных значений порядковых типов в языке Pascal определен так называемый оператор варианта — `case`. Рассмотрим его подробнее.

Оператор варианта

Этот оператор представляет собой естественное расширение условного оператора. Оператор варианта состоит из выражения порядкового типа и нескольких операторов, каждому из которых предшествует список констант того же типа, какого было выражение. Оператор всегда начинается словом `case` и заканчивается словом `end`:

```
case <выражение порядкового типа> of
  константа1: оператор1;
  константа2, константа3, ..., константа10: оператор2;
  константа11..константа15: оператор3;
  else оператор4
end
```

Приведенная схема не есть точное описание оператора; это лишь демонстрация нескольких случаев комбинации констант (полная синтаксическая диаграмма данного оператора слишком громоздка). По-

рядок употребления констант произвольный, но они не должны повторяться. Выполняется оператор так. Сначала вычисляется значение *выражения*, затем среди констант ищется равная значению выражения и выполняется соответствующий оператор. Если соответствующая константа не найдена, то выполнится оператор, стоящий после слова `else`, а если это слово также отсутствует, то не будет делаться ничего и программа перейдет к выполнению следующего оператора.

Приведем примеры нескольких различных операторов варианта:

1)
```
case n mod 7 of
  1: writeln('Понедельник');
  2: writeln('Вторник');
  3: writeln('Среда');
  4: writeln('Четверг');
  5: writeln('Пятница');
  6: writeln('Суббота');
  0: writeln('Воскресенье');
end;
```
2)
```
case c of
  '+': x := x + y;
  '-': x := x - y;
  '*': x := x * y;
  else writeln('error')
end;
```
3)
```
case c of
  'a'..'z', 'A'..'Z': writeln('letter');
  '0'..'9':           writeln('digit')
end;
```

Синтаксис этого оператора опровергает ряд утверждений, справедливых для конструкций языка, рассматривавшихся ранее, а именно следующие: «перед `else` точка с запятой не ставится» и «каждому слову `end` соответствует слово `begin`».

Задачи

1. Выдайте на экран все символы загруженной кодовой таблицы по порядку их номеров, помещая по 60 символов в строке (в первой строке — символы с 0 по 59, во второй — с 60 по 119 и т. д.). На эффект от спецсимволов, таких как звуковой сигнал и перевод строки, не обращайте внимания. Вложенные циклы или несколько последовательных циклов не использовать.

2. Напишите программу, которая будет для любой нажатой клавиши, генерирующей один символ, выдавать ее символьный код (но-

мер). На экране должен быть как нажатый символ, так и его код. Программа заканчивает свою работу тогда, когда какой-либо символ вводится два раза подряд (код последнего символа второй раз не выводится).

Пример диалога программы:

```
A
65
1
49
1
```

Пример входных данных	Пример выходных данных
A 1 1	65 49

3. Напишите программу, которая будет вводить двузначное шестнадцатеричное число и выводить его десятичный аналог.

Пример входных данных	Пример выходных данных
2A	42

4. Выведите на экран последовательность символов:

```
a
ab
abc
abcd
```

и т. д. до строки, заканчивающейся символом «z».

5. Определите, сколько раз в последовательности символов, заканчивающейся точкой, встречаются цифры и сколько — латинские буквы.

На вход программе подается последовательность символов. Признаком окончания последовательности служит точка. Выведите два натуральных числа: количество цифр в последовательности и количество латинских букв.

Пример входных данных	Пример выходных данных
2A+erG23SDdR.	3 8

6. Дана непустая последовательность слов, состоящих из латинских букв. Соседние слова отделены друг от друга запятой, за последним словом следует точка. Определите количество слов, которые начинаются с буквы «а» (как строчной, так и заглавной).

На вход программе подается последовательность слов, разделенных запятыми без пробелов. Последовательность заканчивается точкой. Выведите количество слов в этой последовательности, начинающихся с буквы «а» (как строчной, так и заглавной).

Пример входных данных	Пример выходных данных
`A,erG,adR.`	`2`

7. Для введенного натурального числа k от 1 до 120 напечатайте фразу:

```
Мне k лет
```

Учтите, что при некоторых значениях k слово *лет* надо заменить на слово *год* или *года*. В программе обязательно нужно использовать оператор `case`. В случае автоматической проверки задачи фразу надо выводить в кодировке Windows-1251 (для Borland Pascal программу можно доработать в Блокноте). Соблюдайте регистр при выводе символов и разделяйте слова ровно одним пробелом.

Примеры входных данных	Примеры выходных данных
`15`	`Мне 15 лет`
`21`	`Мне 21 год`

8. Напишите программу, вычисляющую значение арифметического выражения вида: цифра, знак операции (+, − или $*$), цифра. Пользователь вводит без пробелов указанное выражение, например $1+2$ или $7*8$, и программа печатает результат.

Пример входных данных	Пример выходных данных
`7*8`	`56`

9. Дана следующая программа:

```
type direction = (north, east, south, west);
          curs = (forwrd, left, right, back);
var k1, k2: direction;
    p: curs;
    n: integer;
begin
  {считываем номер направления корабля
   0-север, 1-восток, 2-юг, 3-запад}
```

```
  readln(n);
  k1 := direction(n);
  {считываем номер изменения курса
   0-прямо, 1-налево, 2-направо, 3-назад}
  readln(n);
  p := curs(n);
  ...
  writeln('Новый курс ', ord(k2))
end.
```

Корабль шел по курсу `k1` (тип `direction`), ему был дан приказ `p` (`тип curs`). Определите значение курса (тип `direction`), которое получит корабль в результате выполнения приказа.

Замените многоточие на операторы, решающие поставленную задачу.

Пример входных данных	Пример выходных данных
0 2	1

10. Напишите программу, которая будет печатать таблицу истинности логической функции трех переменных, например:

`a or not b and c.`

Урок 9

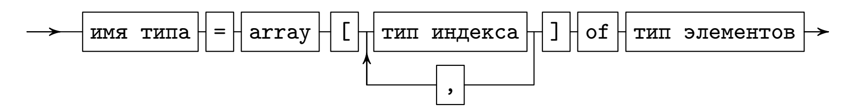
Одномерные массивы

Ранее обсуждались простые типы данных. По-другому они называются *скалярными* типами данных. Каждый из этих типов характеризуется набором множества своих значений (напомним, что это относится в том числе и к вещественным типам, однако перечислить конкретные значения элементов соответствующего множества для того или иного вещественного типа достаточно сложно, для этого необходимо абсолютно точно знать, как именно он реализован в компьютере) и множеством допустимых операций над переменными данного типа. Напомним также, что все скалярные типы, кроме вещественных, являются также *порядковыми* типами.

Из элементов простых типов в языке Pascal можно образовывать составные типы данных, так называемые *структуры данных*. Примером такой структуры является векторный тип данных — одномерный *массив*.

Массив — это составной объект, образованный из элементов (*компонент*) одного и того же типа. Такой тип данных применяется в программировании для обозначения объектов, аналогичных числовым последовательностям в математике, где сразу целая группа чисел обозначается одним именем (чаще всего буквой), а для обращения к каждому отдельному числу данной последовательности используются их *индексы* (номера элементов). В математике это выглядит, например, так: $a_1, a_2, a_3, \ldots, a_n$.

Для описания подобных объектов в программировании полезно предварительно ввести соответствующий тип в разделе описания типов. Описание типа данных *массив* производится согласно следующей синтаксической диаграмме:



Данная диаграмма соответствует не только одномерным, но и многомерным массивам.

Все компоненты массива (то есть составляющие его элементы) нумеруются элементами упорядоченного множества индексов, при-

надлежащих к одному из порядковых типов. Порядковые типы могут быть различными, но чаще всего на практике для этого используется ограниченный тип (диапазон) целых чисел, например `1..100`, то есть фактически на месте *типа индекса* обычно стоит следующая конструкция:

→ `минимальное значение индекса` → `..` → `максимальное значение индекса` →

В качестве типа индекса помимо ограниченных типов могут быть непосредственно указаны типы `byte`, `char`, `boolean` и введенные программистом перечислимые типы. В последних трех случаях индексами будут являться не числа, а символы или соответствующие константы перечислимого типа. Такие возможности иногда оказываются чрезвычайно полезными, см., например, задачу 6 к материалам этого урока. Типы `integer`, `word` и `longint`, не используются в аналогичных целях лишь потому, что в этом случае количество описанных элементов массива превышает возможности по резервированию памяти компилятора или операционной системы. Так, в языке Borland Pascal на все переменные, описанные в одном блоке (в частности, на все глобальные переменные), отводится 64 Кб, а именно столько однобайтовых значений содержал бы массив, проиндексированный числами типа `integer` или `word`, и никакой другой переменной описать уже было бы невозможно, то есть обработать подобный массив нельзя[1]. В языке Delphi, согласованном с моделью памяти Windows, это ограничение на размер отводимой под данные памяти снято, но тип `integer` в Delphi определяет уже $2^{32} \approx 4 \times 10^9$ различных значений, что слишком много для 32-битной операционной системы.

Тип же самих элементов может быть любым, в том числе и составным. Количество элементов массива называется его *размерностью*. Несложно подсчитать, что размерность массива равна

$$\textit{максимальное значение индекса} - \textit{минимальное значение индекса} + 1.$$

Приведем пример структуры данных для описания, например, числовой последовательности, состоящей не более чем из 100 элементов:

```
const nmax = 100;
type aa = array[1..nmax] of real;
var a: aa;
```

[1] На самом деле переменным отводится на несколько байт меньше, чем 64 Кб, поэтому и просто завести такой массив не удастся.

Итак, мы создали массив, состоящий из 100 вещественных чисел. Меняя значение константы nmax, мы можем изменять количество элементов в массиве.

Индексы элементов массива могут начинаться с любого целого числа, в том числе и отрицательного, например:

```
type bb = array[-5..3] of boolean;
```

Массивы данного типа будут содержать 9 логических значений, пронумерованных от −5 до 3.

Над переменными типа массив возможна только операция присваивания, то есть содержимое одного массива может быть присвоено содержимому другого массива того же самого типа. Например, если мы добавим к приведенному выше описанию переменной a переменную b того же типа aa, то в программе будет возможно использовать следующий оператор присваивания:

```
b := a;
```

Очевидно, что для того чтобы этот оператор имел смысл, значения элементов массива a уже должны быть заданы, например введены с клавиатуры.

С самими элементами массива можно осуществлять все те действия, которые допустимы с обыкновенными переменными соответствующего типа (real, integer и т. д.) Чтобы обратиться в программе к конкретному элементу массива, после имени переменной типа массив в квадратных скобках нужно поставить индекс элемента массива. Это может быть константа, входящая в диапазон констант, указанный при описании, это может быть переменная того же самого порядкового типа (если в качестве индексов используется конкретный диапазон целых значений, то переменная может принадлежать любому целому типу), наконец, это может быть произвольное выражение, значение которого также принадлежит указанному типу, например a[5], a[i], a[i + 1], a[2 * k - 1]. При использовании переменных для обозначения индекса их значения к моменту использования должны быть определены, а в случае арифметических выражений их результат не должен выходить за границы массива (минимальное и максимальное значения индекса).

Чаще всего массив обрабатывается в цикле for. Так, для того чтобы считать с клавиатуры описанный выше массив a, нужно написать следующее:

```
readln(n);
for i := 1 to n do
  read(a[i]);
readln;
```

Здесь n — реальный размер массива, который часто является параметром задачи, он не должен превышать значения nmax, использованного при описании этого массива.

При запуске соответствующей программы значения элементов массива можно набивать в одну строчку через пробел, а можно — каждое в отдельной строке. Приведенный во второй строке оператор readln не является обязательным, он лишь отражает тот факт, что после ввода значений будет обязательно нажата клавиша перевода строки Enter. Отсутствие такого оператора не повлияет на правильность выполнения данного фрагмента программы, но может повлиять на дальнейшие операции ввода. В частности, если в данном месте readln отсутствует, но зато именно он используется в качестве последнего оператора программы, для того чтобы она не заканчивалась, пока пользователь не ознакомится с результатами ее работы и не нажмет клавишу Enter, то ожидаемого эффекта от наличия этого оператора не произойдет. Связано это с тем, что все данные, вводимые с клавиатуры, попадают в так называемый «буфер клавиатуры», а операторы readln и readln «забирают» оттуда данные. Если единственный оператор readln находится в конце программы, то он просто заберет из буфера клавиатуры символ перевода строки, так как последний заведомо попадет туда при вводе элементов массива. Соответственно программа сразу же завершит свою работу — возможности проанализировать результат у пользователя не будет.

По-другому сказанное выше можно сформулировать так: грамотно написанный ввод данных должен сразу обрабатывать все входные символы, в частности, возникающие в результате нажатия клавиши перевода строки. Именно такой подход и был продемонстрирован в приведенном примере описания ввода значений элементов массива.

Аналогично производится печать элементов массива. Но просто заменить read на write здесь недостаточно. Для того чтобы печатаемые значения не сливались между собой, надо явным образом вставлять между ними разделитель — пробел или перевод строки. Приведем два возможных способа распечатки массива:

1) `for i := 1 to n do write(a[i], ' ');`
   `writeln;`

2) `for i := 1 to n do writeln(a[i]);`

На первый взгляд второй способ может показаться более простым и удобным, но это далеко не всегда так. Результат работы такой программы зачастую неудобно, а то и просто невозможно анализировать. Ведь каждый элемент массива будет располагаться в отдельной строке, следовательно, мы не сможем увидеть более 25 (или 48) элемен-

тов одновременно. Кроме того, очень часто массив требуется распечатать дважды, чтобы сравнить состояние массива до обработки и в результате его обработки. В этом случае сравнение состояний массива гораздо удобнее проводить, если они напечатаны в двух соседних строках, а элементы выровнены по столбцам, то есть добавлена еще и печать по формату (указано количество позиций, которое должно отводиться на печать одного элемента, например `a[i]:5`).

Вводить значения элементов с клавиатуры не всегда удобно. Более того, часто при отладке программы нам надо сделать так, чтобы она работала на произвольных массивах. В этом случае очень удобно задавать значения элементов массива случайным образом, с использованием встроенного в язык программирования так называемого *генератора псевдослучайных чисел*. Для этого используются функция `random` и процедура `randomize`. Если использовать первую из них без параметров, то при очередном обращении она будет выдавать в качестве результата некоторое псевдослучайное вещественное число из диапазона $[0; 1)$. Процедура `randomize` предназначена для задания первого значения в данной последовательности. Обращаться к процедуре `randomize` имеет смысл только один раз — в начале работы программы. Для получения целых случайных чисел из диапазона $[0; n-1]$ используется вызов функции `random` с параметром `n`: `random(n)`. Пример заполнения массива `m`, состоящего из `n` целочисленных элементов, случайными десятичными цифрами:

```
randomize;
for i := 1 to n do m[i] := random(10);
```

Обратите внимание на то, что ни один из элементов этого массива не может оказаться равным 10, но элементы могут быть равны нулю.

Приведем пример работы с массивом. Пусть заданы массив `a` введенного ранее типа `aa` и его реальная размерность `n` (под словом «задан» здесь и далее мы будем понимать, что массив не только описан, но и значения его элементов определены тем или иным способом). Требуется сдвинуть элементы массива на одну позицию влево, а первый элемент поставить на последнее место. Следующий фрагмент решает данную задачу:

```
c := a[1];
for i := 1 to n - 1 do
  a[i] := a[i + 1];
a[n] := c;
```

Переменная `c` здесь должна быть того же типа, что и элементы массива, в нашем случае — `real`.

Задачи

1. Заполните числовой массив так, чтобы

а) значениями элементов оказались случайные целые числа от a до b включительно;

б) значениями элементов были случайные вещественные числа от 0 до 10;

в) значения элементов совпадали с их индексами;

г) значения элементов были равны квадратам индексов.

В последнем случае на вход программе подаются два целых числа a и b, по модулю не превосходящие 100, $a \leqslant b$. Требуется вывести часть массива с индексами от a до b, элементы которого равны квадратам своих индексов.

Пример входных данных	Пример выходных данных
`-1 3`	`1 0 1 4 9`

2. Напишите программу, которая будет циклически сдвигать заданный массив на один элемент вправо, последний элемент при этом должен оказаться на первом месте.

На вход программе сначала подается значение $n \leqslant 100$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30 000. Выдайте значения элементов массива после выполнения указанной операции.

Пример входных данных	Пример выходных данных
`5` `5 4 3 2 1`	`1 5 4 3 2`

3. Найдите максимальный и минимальный элементы в массиве и поменяйте их местами.

На вход программе сначала подается значение $n \leqslant 100$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30 000. Выдайте значения элементов массива после выполнения указанной операции.

Пример входных данных	Пример выходных данных
`5` `5 4 3 2 1`	`1 4 3 2 5`

4. Распечатайте те элементы массива, которые равны сумме двух своих соседей. Первый и последний элемент имеют только по одному соседу, поэтому искомыми быть не могут.

На вход программе сначала подается значение $n \leqslant 100$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 10 000. Выдайте значения искомых элементов массива в том же порядке, в каком они располагались во входных данных.

Пример входных данных	Пример выходных данных
`5` `1 5 4 6 2`	`5 6`

5. На вход программе подается последовательность чисел от 1 до 9, заканчивающаяся нулем. Всего будет введено не более 100 000 чисел. Подсчитайте в этой последовательности количество единиц, количество двоек, количество троек и т. д. и выдайте результат. В выходных данных всегда должно быть 9 чисел.

Пример входных данных	Пример выходных данных
`1 1 4 1 5 8 6 3 5 1 0`	`4 0 1 1 2 1 0 1 0`

6. Текст на английском языке запишите в массив `a[1..1000] of char`. Помимо английских букв в нем могут встречаться пробелы и знаки препинания. В массиве `b['A'..'Z'] of integer` получите сведения о том, сколько каких букв встречается в этом тексте. При подсчете строчные и прописные буквы не различать.

На вход программе сначала подается значение $n \leqslant 1000$ — количество символов в тексте. В следующей строке входных данных расположены сами символы (без разделителей). Выдайте 26 чисел — значения элементов массива `b`.

Пример входных данных	Пример выходных данных
`12` `Hello world!`	`0 0 0 1 1 0 0 1 0 0 0 3 0 0 2 0 0 1 0 0 0 0 1 0 0 0`

7. Подсчитайте за один проход массива, сколько его элементов равны максимальному элементу.

На вход программе сначала подается значение $n \leqslant 100$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30 000. Выдайте количество искомых элементов массива.

Пример входных данных	Пример выходных данных
8 4 3 5 2 5 1 3 5	3

8. В массиве, заполненном произвольными целыми числами, найдите два числа, произведение которых максимально. Вложенные циклы не используйте.

На вход программе сначала подается значение $n \leqslant 10\,000$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30 000. Выдайте искомые числа в порядке неубывания.

Примеры входных данных	Примеры выходных данных
5 4 3 5 2 5	5 5
5 -4 3 -5 2 5	-5 -4

9. На вход программе сначала подается значение $n \leqslant 100$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30 000. Распечатайте только те значения элементов массива, которые встречаются в нем ровно один раз. Элементы следует распечатывать в том порядке, в котором они встречаются в массиве.

Пример входных данных	Пример выходных данных
8 4 3 5 2 5 1 3 5	4 2 1

10. На вход программе сначала подается значение $n \leqslant 100$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30 000. Распечатайте только те значения элементов массива, которые встречаются в нем более одного раза, при этом каждое

найденное значение должно быть распечатано только один раз. Элементы следует распечатывать в том порядке, в котором они встречаются в массиве.

Пример входных данных	Пример выходных данных
8 4 3 5 2 5 1 3 5	3 5

Задачи повышенной сложности

1. По введенному натуральному числу k, не превосходящему 100 000, выдайте k-е по счету простое число. Используйте массив для запоминания уже найденных простых чисел или для хранения «решета Эратосфена» в случае его испльзования.

Примеры входных данных	Примеры выходных данных
1	2
4	7

2. Определите, сколько раз число $n!$ (факториал числа n) нацело делится на натуральное число k ($1 < n, k < 2^{31}$).

На вход программе подаются целые числа n и k, выведите число, указывающее, сколько раз $n!$ нацело делится на k.

Примеры входных данных	Примеры выходных данных
6 3	2
6 12	2

3. Задан массив, содержащий несколько нулевых элементов. Требуется «сжать» его, переместив нулевые элементы в правую часть массива. Порядок ненулевых элементов относительно друг друга не менять. Нужно изменить исходный массив, а не распечатать ненулевые элементы или записать их в дополнительный массив.

На вход программе сначала подается значение $n \leqslant 10\,000$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30 000. Распечатайте «сжатый» массив.

Пример входных данных	Пример выходных данных
8 4 0 5 0 3 0 0 5	4 5 3 5 0 0 0 0

4. Даны массив натуральных чисел a_1, a_2, ..., a_n и натуральные числа k и m. Укажите минимальное значение i, для которого $a_i + a_{i+1} + \ldots + a_{i+k} = m$ (то есть сумма $k+1$ подряд идущих элементов массива равна m). Если такого значения нет, то выведите 0. Вложенные циклы не использовать.

На вход программе сначала подаются значения n, k и m ($m \leqslant 10\,000$, $0 < k < n \leqslant 30\,000$; n — количество элементов в массиве). В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 100.

Пример входных данных	Пример выходных данных
`8 1 5` `4 0 5 0 3 2 3 3`	`2`

5. В одномерном массиве, заполненном произвольными целыми числами, за один проход найдите непрерывный кусок, сумма чисел в котором максимальна. Фактически требуется найти такие i и j ($i \leqslant j$), что сумма всех элементов массива от a_i до a_j включительно будет максимальна.

На вход программе сначала подается значение $n \leqslant 10^5$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30 000. Выдайте пару искомых значений индексов. Если таких пар несколько, то j должно быть минимально возможным, а при равных j значение i должно быть максимально возможным.

Примеры входных данных	Примеры выходных данных
`5` `-1 2 3 -2 2`	`2 3`
`7` `2 -2 3 -1 5 -2 7`	`3 7`

6. Заданы два неупорядоченных массива целых чисел, по модулю не превосходящих 10 000. Будем рассматривать их как множества с повторяющимися элементами. Не используя дополнительной памяти, требуется распечатать пересечение двух множеств. Так, массивы $\{2, 1, 2, 3\}$ и $\{2, 4, 2, 2\}$ имеют пересечение $\{2, 2\}$. По окончании работы программы сами массивы должны быть такими же, какими они были после заполнения.

На вход программе сначала подается значение $n \leqslant 100$ — количество элементов в первом массиве. В следующей строке входных

данных расположены элементы первого массива. Далее на вход программе подается значение $m \leqslant 100$ — количество элементов во втором массиве. В следующей строке входных данных расположены элементы второго массива. Элементы в обоих массивах — целые числа, по модулю не превосходящие 10 000.

Выдайте общие элементы данных массивов в том порядке, в котором они встречаются в первом массиве.

Пример входных данных	Пример выходных данных
5 2 1 2 3 4 4 2 4 2 2	2 2 4

7. Напишите эффективную программу, которая будет циклически сдвигать заданный массив на k элементов вправо. Дополнительные массивы и рекурсию не используйте.

На вход программе сначала подаются значения $n \leqslant 100$ — количество элементов в массиве и $k \leqslant 100$. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30 000. Выдайте значения элементов массива после выполнения указанной операции.

Работа программы не должна зависеть от значения k.

Пример входных данных	Пример выходных данных
5 3 5 4 3 2 1	3 2 1 5 4

8. Числовая последовательность называется *пилообразной*, если каждый ее элемент (кроме первого и последнего) либо больше обоих своих соседей, либо меньше обоих соседей. Например, последовательность 1, 2, 1, 3, 2 является пилообразной, а 1, 2, 3, 1, 2 — нет, поскольку $1 < 2 < 3$. Любая последовательность из одного элемента является пилообразной. Последовательность из двух элементов является пилообразной, если ее элементы не равны.

Дана последовательность. Требуется определить, какое наименьшее количество ее элементов нужно вычеркнуть, чтобы оставшаяся последовательность оказалась пилообразной.

На вход программе сначала подается значение n ($1 \leqslant n \leqslant 100\,000$) — количество членов последовательности. Во второй строке записаны n натуральных чисел, не превосходящих 10 000 — члены последовательности. Выведите одно число — минимальное количество элементов, которые необходимо вычеркнуть.

Примеры входных данных	Примеры выходных данных
5 1 2 3 1 2	1
5 1 2 1 3 2	0
5 1 2 3 4 5	3
5 1 1 2 1 1	2

9. В одномерном массиве, заполненном произвольными целыми числами, для каждого элемента найдите номер ближайшего справа элемента, меньшего данного. Задачу следует решить за один проход массива.

На вход программе сначала подается значение $n \leqslant 10^5$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30 000. Выдайте n искомых значений индексов. Если у элемента нет справа элемента, меньшего данного, то для него в качестве индекса выдайте 0.

Пример входных данных	Пример выходных данных
6 2 3 1 5 6 4	3 3 0 6 6 0

10. В одномерном массиве, заполненном произвольными целыми числами, найдите два таких числа, для которых разность между правым и левым из них будет максимальна: $\max\limits_{0<i<j\leqslant n}(a[j]-a[i])$. Задачу следует решить за один проход массива.

На вход программе сначала подается значение $n \leqslant 10^5$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30 000. Выдайте значения двух искомых элементов.

Пример входных данных	Пример выходных данных
5 6 2 4 5 1	2 5

Урок 10

Двумерные массивы (матрицы)

Как уже было сказано выше, элементами массива могут быть данные любого из известных типов, следовательно, и сам массив. Таким образом, допустимо, например, следующее описание:

```
const n = 10; m = 20;
type aa2 = array[1..n] of array[1..m] of real;
var a: aa2;
```

Это описание задает таблицу (матрицу) вещественных чисел, состоящую из 10 строк и 20 столбцов. Каждый элемент таблицы имеет два индекса. Обратиться к конкретному элементу таблицы из программы можно, например, следующим образом: `a[5][3]`, что соответствует элементу, стоящему в пятой строке и третьем столбце.

Более распространенным является другое описание двумерных массивов. В нашем примере оно будет выглядеть так:

```
type bb2 = array[1..n, 1..m] of real;
var b: bb2;
```

При этом описании обращение к конкретному элементу выглядит, например, так: `b[5, 3]`.

Обработка двумерных массивов производится с помощью двух вложенных циклов. Например, считать элементы массива и распечатать их в виде таблицы можно следующим образом:

```
for i := 1 to n do
  for j := 1 to m do
    read(b[i, j]);
readln;
for i := 1 to n do
begin
  for j := 1 to m do
    write(b[i, j]:6:1);
  writeln
end;
```

Приведенные способы описания распространяются и на массивы большей размерности.

Элементы двумерных массивов в памяти компьютера располагаются «по строкам», то есть сначала расположены все элементы первой строки (первый индекс фиксирован и равен своему минимальному значению), затем второй и т. д. Таким образом, при равноправном расположении вложенных циклов для обработки многомерного массива сначала следует организовать цикл по первому индексу, потом по второму и т. д. (так, как это и сделано в приведенном выше примере).

Особый интерес представляют квадратные матрицы (таблицы), — двумерные массивы, у которых оба измерения совпадают, например:

```
a: array [1..10, 1..10] of integer;
```

Рассмотрим, как эффективно обрабатывать различные части матриц. Проведем в матрице так называемую главную диагональ — линию, соединяющую левый верхний и правый нижний угол квадратной матрицы. Пометим все элементы цифрами 0, 1 или 2 в зависимости от положения элемента матрицы относительно главной диагонали:

0	1	1	1	1
2	0	1	1	1
2	2	0	1	1
2	2	2	0	1
2	2	2	2	0

Элементы, стоящие на главной диагонали матрицы, обозначены цифрой 0. У произвольного элемента квадратной матрицы `a[i,j]` i — это номер строки, а j — это номер столбца. Тогда принадлежность элемента главной диагонали описывается условием $i = j$. А для обработки всех элементов, находящихся на главной диагонали, достаточно одного цикла:

```
for i := 1 to n do
  ...a[i, i]...
```

Элементы верхнего треугольника, помеченные цифрой 1, лежат правее главной диагонали. Для их обработки можно написать цикл, который также не будет содержать никаких условий:

```
for i := 1 to n - 1 do {это цикл по всем строкам}
  for j := i + 1 to n do
{цикл по столбцам: от диагонали до n}
    ...a[i, j]...
```

Наконец, элементы нижнего треугольника, помеченные цифрой 2, описываются так:

```
for i := 2 to n do
  for j := 1 to i - 1 do
    ...a[i, j]...
```

Задачи

1. а) Заполните и выведите двумерный массив $n \times n$ символами пробел и «*» так, чтобы получилась «снежинка». На вход программе подается нечетное значение n. Например, для $n = 5$ массив должен быть заполнен так:

*		*		*
	*	*	*	
*	*	*	*	*
	*	*	*	
*		*		*

б) Заполните и выведите двумерный массив размера 8×8 так, чтобы получилась шахматная доска. Белые клетки заменяйте пробелами, а черные — символами «*»:

*		*		*		*	
	*		*		*		*
*		*		*		*	
	*		*		*		*
*		*		*		*	
	*		*		*		*
*		*		*		*	
	*		*		*		*

2. Разделим квадратную матрицу диагональю, соединяющую правый верхний элемент с левым нижним. Такую диагональ обычно на-

зывают «побочной»:

4	4	4	4	3
4	4	4	3	5
4	4	3	5	5
4	3	5	5	5
3	5	5	5	5

Распечатайте в виде треугольной таблицы элементы матрицы, стоящие на местах, обозначенных цифрами 3 и 4.

На вход программе сначала подается значение $n \leqslant 20$ — размер квадратной матрицы. В следующих n строках входных данных расположены сами элементы матрицы — натуральные числа, меньшие 100. Выведите требуемые элементы, выравнивая их по столбцам.

Пример входных данных	Пример выходных данных
4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	1 2 3 4 5 6 7 9 10 13

3. В кинотеатре n рядов по m мест в каждом. В соответствующем двумерном массиве хранится информация о проданных билетах на определенный сеанс (единицы означают, что на данные места билеты уже проданы, нули — что данные места еще свободны). Поступил запрос на продажу k билетов на соседние места в одном ряду. Определите, можно ли удовлетворить такой запрос.

В первой строке входных данных находятся числа n, m, $k \leqslant 100$. В следующих n строках входных данных расположены по m чисел (0 и 1), разделенных пробелами. Выведите `YES` или `NO` в зависимости от ответа на вопрос задачи.

Примеры входных данных	Примеры выходных данных
3 4 2 0 1 0 1 1 0 0 1 1 1 1 1	YES
3 3 3 0 1 0 1 0 0 1 1 1	NO

4. Треугольник Паскаля строится следующим образом. Первая строка состоит из одной единицы. Каждая следующая содержит на одно число больше, чем предыдущая. Первое и последнее из этих чисел равны 1, а все остальные вычисляются как сумма числа, стоящего в предыдущей строке над ним, и числа, стоящего в предыдущей строке слева от него.

По введенному $n \leqslant 30$ выведите n первых строк треугольника Паскаля.

Пример входных данных	Пример выходных данных
5	1 1 1 1 2 1 1 3 3 1 1 4 6 4 1

5. Во входных данных описан план комнаты: сначала количество строк n, затем — количество столбцов m ($1 \leqslant n \leqslant 20$, $1 \leqslant m \leqslant 20$). Затем записано n строк по m чисел в каждой — количество килограммов золота, которое лежит в данной клетке (число от 0 до 50). Далее записано число x — сколько клеток обошел мудрец. Далее записаны координаты этих клеток (координаты клетки — это два числа: первое определяет номер строки, второе — номер столбца), верхняя левая клетка на плане имеет координаты $(1, 1)$, правая нижняя — (n, m).

Выведите количество килограммов золота, которое собрал мудрец. В задаче *не* гарантируется, что мудрец не проходил по одной и той же клетке более одного раза.

Пример входных данных	Пример выходных данных
3 5 1 2 3 4 5 0 9 8 7 6 1 2 1 4 1 4 1 1 2 2 3 1 2 2	11

6. По введенным значениям n, m ($1 \leqslant n \leqslant 20$, $1 \leqslant m \leqslant 20$) заполните массив размерностью $n \times m$ числами от 1 до mn, расположив их горизонтальной «змейкой» так, как показано в примере.

Пример входных данных	Пример выходных данных
3 5	1 2 3 4 5 10 9 8 7 6 11 12 13 14 15

7. По введенным значениям n, m ($1 \leqslant n \leqslant 20$, $1 \leqslant m \leqslant 20$) заполните массив размерностью $n \times m$ числами от 1 до mn, расположив их по спирали, закрученной по часовой стрелке, так, как показано в примере.

Пример входных данных	Пример выходных данных
4 4	1 2 3 4 12 13 14 5 11 16 15 6 10 9 8 7

8. Дан квадратный массив. Требуется повернуть его на 90° по часовой стрелке (результат можно записать в другой массив).

На вход программе сначала подается значение $n \leqslant 20$ — размер массива. В следующих n строках входных данных расположены сами элементы массива — натуральные числа, меньшие 100. Выведите массив, полученный после поворота исходного.

Пример входных данных	Пример выходных данных
4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	13 9 5 1 14 10 6 2 15 11 7 3 16 12 8 4

9. В двумерном массиве размерностью $n \times m$, все элементы которого различны, требуется найти такие элементы, которые одновременно являются минимальными в своей строке и максимальными в своем столбце.

В первой строке входных данных находятся натуральные числа n, m, $k \leqslant 100$. В следующих n строках входных данных расположены по m натуральных чисел, не превосходящих 10 000. Выведите пары индексов искомых элементов, каждую в отдельной строке. Нумерация строк и столбцов начинается с единицы.

Если искомых элементов нет, то выведите 0.

Примеры входных данных	Примеры выходных данных
3 4 1 2 3 4 5 6 7 8 9 10 11 12	3 1
2 2 3 1 2 4	0

10. На шахматной доске расположены несколько слонов и ладей. По условным буквенным обозначениям фигур и их координатам определите, сколько свободных полей шахматной доски *не* находятся под боем ни одной из этих фигур.

Шахматная доска описывается в восьми строках входных данных. Первые восемь символов каждой из этих строк описывают состояние соответствующей горизонтали: символ B (заглавная латинская буква) означает, что в клетке стоит слон, символ R — ладья, символ * — что клетка пуста.

Выведите количество пустых клеток, которые не бьет ни одна из фигур.

Примеры входных данных	Примеры выходных данных
******** *RB***** ******** ******** ******** ******** ******** ********	47
RRRRRRRR BBBBBBBB RRRRRRRR BBBBBBBB RRRRRRRR BBBBBBBB RRRRRRRR BBBBBBB*	0

Урок 11

Строки

Тип `string` (строка) не входит в стандартный язык Pascal, но он прочно вошел во все известные компиляторы с этого языка. Причин, по которым он был добавлен в язык, несколько. При изучении массивов говорилось, что единственной доступной операцией над подобными составными объектами является операция присваивания. Для последовательностей символов этой операции явно не достаточно. Кроме того, при обработке массивов символов требуются дополнительные операции, которые часто применяются при решении совершенно различных задач. Эти операции были оформлены в виде стандартных процедур и функций. Если говорить о том, какую именно информацию удобно хранить в переменных строкового типа, то в первую очередь следует упомянуть последовательности символов, обозначающие, например, фамилии людей или географические названия. Именно для подобных данных над строками были введены операции сравнения. Кроме того, одно слово любого текста (последовательность символов, окруженная разделителями) также удобно анализировать в переменной строкового типа. В строковой переменной можно хранить и содержимое одной строки большого текста. Сам же текст целиком удобнее хранить в файле, массиве символов или массиве строк.

Значениями данного типа в Borland Pascal являются различные последовательности символов длиной от 0 до 255 символов. При описании строковой переменной в квадратных скобках может быть указан ее максимальный размер, не превосходящий 255. Если такой размер опущен, то считается, что длина строки может достигать 255 символов. Указание размера строки влияет только на распределение памяти для хранения соответствующих переменных. Все переменные строковых типов совместимы между собой при выполнении различных операций над строками.

Примеры описания строк:

```
var s1, s2: string;
  name: string[20];
  group: string[3];
  txt: array[1..25] of string[80];
```

Строку почти всегда можно рассматривать как массив символов, то есть обращаться из программы к отдельным символам строки: `s1[1]`, `s2[i]`, `txt[1][10]` (в последнем случае имеется в виду десятый символ первой строки). Нумерация символов в строке всегда начинается с 1. Однако присваивание отдельным элементам строки конкретных символьных значений может не привести к желаемому результату. Чтобы понять, почему это иногда происходит, необходимо рассмотреть принципы организации компилятором Borland Pascal хранения строковых переменных в памяти компьютера. Дело в том, что для хранения любой строки отводится на 1 байт больше, чем указано при ее описании (256 байт для строк, размерность которых не указана явно), причем этот вспомогательный байт располагается в самом начале строки, фактически это ее нулевой байт. В нем хранится текущая реальная длина строки. Собственно говоря, такой механизм и накладывает максимальные ограничения на возможную длину строк в Borland Pascal, так как максимальное число, представимое в одном байте, равно 255. Значение этого байта автоматически изменяется программой при выполнении стандартных операций над строками: считывании, присваивании строковой переменной строкового выражения, обращении к стандартным процедурам и функциям, работающим со строковыми выражениями. Рассмотрим на примере, как изменяются содержимое памяти и реальное значение строки при выполнении тех или иных операций:

Операция	Содержимое памяти, выделенной строковой переменной s							Значение s
var s: string [6];	s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	
s := '';	0							''
s := 'Yes';	3	Y	e	s				'Yes'
s[4] := '!';	3	Y	e	s	!			'Yes'
s := s + 'no'	5	Y	e	s	n	o		'Yesno'
s[4] := '!';	5	Y	e	s	!	o		'Yes!o'

На этом примере можно понять, что операция присваивания значения элементу строки, индекс которого превосходит текущую длину строки, не является корректной и фактически не оказывает влияния на результат. Поэтому применения подобных конструкций при программировании следует избегать, стараясь обходиться стандартными операциями над строками. Рассмотрим их подробнее.

В отличие от массивов над строковыми переменными определены операции считывания и печати. К строкам применима операция

сложения. При сложении двух строк к концу первой из них приписывается вторая (по-другому эту операцию называют *конкатенацией* строк). Над строками реализованы и все операции сравнения. Сравнение строк между собой производится согласно так называемому *лексикографическому порядку*. Строка `s1` считается меньше строки `s2`, если существует такая позиция k, что символы, стоящие в указанных строках на позициях с первой по $(k-1)$-ю, совпадают, а `s1[k] <` `< s2[k]`, или такой позиции не существует, но строка `s1` короче строки `s2`. Например:

`'abracadabra' < 'baobab'` (здесь $k=1$);

`'abba' < 'abra'` (здесь $k=3$);

`'abb' < 'abba'` (здесь строки сравнивались по длине).

Приведем также основные процедуры и функции для работы со строками:

`length(s)` — функция, определяющая реальную длину строки;

`copy(s, i, n)` — функция, выделяющая из строки `s` подстроку длиной `n`, начиная с `i`-го символа;

`delete(s, i, n)` — процедура, удаляющая из строки `s` `n` символов, начиная с `i`-го символа;

`pos(s1, s)` — функция, в качестве результата выдающая номер позиции в строке `s`, с которой начинается подстрока `s1`; если подстрока не найдена, то результат равен 0;

`insert(s1, s, i)` — процедура, вставляющая в строку `s` подстроку `s1` перед символом с номером `i`;

`val(s, n, i)` — процедура, переводящая строку `s` в число (вещественное или целое, согласно типу переменной `n`), если строка `s` не является изображением числа соответствующего типа по правилам Borland Pascal, то значение переменной `i` будет отлично от 0, при удачной конвертации значение `i` равно 0;

`str(i, s)` — процедура, переводящая число в его строковое представление, у числа можно указать формат, аналогично тому как это делается в операторе `write`.

В языке Delphi, помимо паскалевских «коротких» строк, реализованы и динамические «длинные» строки, размер которых ограничен только возможностями операционной системы. Описываются эти строки так же, как и короткие, с помощью слова `string`. Память под такие переменные выделяется по мере необходимости в процессе выполнения программы. Обращение к несуществующим элементам такой строки, то есть к элементам, индекс которых превосходит текущую длину строки, эквивалентно выходу за границу массива (с короткими строками в пределах 255 символов это не так).

Какие именно строки имеются в виду, определяется с помощью директивы компилятора `H`. Так, директива `{$H+}`, записанная в начале программы, означает, что строки будут «длинные», а `{$H-}` — короткие.

Кроме того, в модуле `SysUtils` языка Delphi реализовано много дополнительных (по отношению ко стандартным паскалевским) функций над строками. Наиболее часто употребляются удобные функции перевода чисел в строки и обратно. Приведем некоторые из них:

`IntToStr(i)` — функция, переводящая целое число в строку;

`StrToInt(s)` — функция, переводящая строку в целое число;

`FloatToStrF(s, ffFixed, d, p)` — функция, переводящая вещественное число в строку по формату `:d:p`, аналогичному форматной печати;

`StrToFloat(s)` — функция, переводящая строку в вещественное число.

Задачи

1. В строке имеется несколько слов, разделенных одним или несколькими пробелами. Требуется убрать из текста лишние пробелы: два и более пробелов подряд, а также все пробелы в начале и в конце строки.

На вход программе подается строка, состоящая не более чем из 255 символов. Выведите преобразованную строку.

Пример входных данных	Пример выходных данных
␣один␣␣два␣␣␣три␣␣	один␣два␣три

В примере пробельные символы изображаются с помощью ␣.

2. Фраза называется палиндромом (перевертышем), если после удаления пробелов и замены всех букв на заглавные она читается одинаково, как слева направо, так и справа налево. Требуется определить, является ли введенная фраза палиндромом.

На вход программе подается строка, состоящая не более чем из 255 символов. Выведите `YES` или `NO` в зависимости от ответа на вопрос задачи.

Примечание. Для русских букв, записанных в кодировке Windows-1251, можно считать, что коды заглавных и строчных букв отличаются на 32.

Примеры входных данных	Примеры выходных данных
А роза упала на лапу Азора	YES
А собака босса	NO

3. Слово называется палиндромом, если оно читается одинаково как слева направо, так и справа налево. Требуется определить, какое минимальное количество букв надо добавить ко входному слову справа, чтобы оно стало палиндромом.

На вход программе подается строка, состоящая не более чем из 255 символов. Выведите искомое число.

Примеры входных данных	Примеры выходных данных
`abcd`	`3`
`abb`	`1`

4. Строка состоит из двух слов. Поменяйте эти слова местами.

На вход программе подается строка, состоящая не более чем из 255 символов. Строка содержит ровно один пробел, разделяющий два слова. Выведите преобразованную строку, по-прежнему разделяя слова ровно одним пробелом.

Пример входных данных	Пример выходных данных
`abcd ef`	`ef abcd`

5. Дана строка, являющаяся параграфом в тексте. Текст необходимо отформатировать так, чтобы длина каждой строки не превосходила числа m, слова при этом не разрывать.

На вход программе сначала подается число m, $0 < m \leqslant 255$. В следующей строке находится исходный текст. Длина слов в нем не превышает m, слова разделены ровно одним пробелом. Выведите разбиение этого текста на строки длиной не более чем m символов (слово переносится на следующую строку, только если в текущей строке его разместить уже невозможно). Новая строка не должна начинаться с пробела.

Пример входных данных	Пример выходных данных
`7` `один два три четыре`	`один` `два три` `четыре`

6. Определите, сколько букв содержит самое длинное слово во введенной строке символов.

На вход программе подается строка, состоящая не более чем из 255 символов. Слова разделяются одним или несколькими пробелами.

Выведите искомое число.

Примеры входных данных	Примеры выходных данных
`abcd ef`	`4`
`Hello world!`	`6`

7. Дана строка, содержащая по крайней мере один неведущий пробел, за которым следует отличный от пробела символ. За счет изменения размеров групп пробелов внутри строк (количества пробелов между словами) добейтесь того, чтобы в начале и в конце каждой из строк пробелы отсутствовали. Количество пробелов в разных группах внутри одной строки должно различаться не более чем на единицу. Количество символов в строке должно остаться неизменным.

На вход программе подается строка, состоящая не более чем из 255 символов. Выведите преобразованную строку. Если количество пробелов между словами различно, то сначала должны идти группы пробелов минимального размера, а затем — на единицу большего размера.

Пример входных данных	Пример выходных данных
`␣один␣␣два␣␣␣три␣`	`один␣␣␣два␣␣␣␣три`

8. Напишите модификацию функции `pos`, которая находит все вхождения искомой подстроки в данной строке. В первой строке вводится исходная строка, в следующей — подстрока. В качестве ответа выведите в порядке возрастания все такие числа i, что, начиная с i-го символа, начинается некоторое вхождение подстроки в исходную строку.

На вход программе сначала подается строка, состоящая не более чем из 255 символов, а затем искомая подстрока, длина которой не превышает длину первой строки. Выведите число 0, если искомая подстрока не найдена, или искомые номера символов, с которых начинается подстрока в строке.

Примеры входных данных	Примеры выходных данных
`fff` `ff`	`1 2`
`abcd` `bac`	`0`

9. Дана последовательность символов, имеющая следующий вид: $p_1 * p_2 * p_3 * \ldots * p_n$, где p_i — цифра. Вычислите значение выражения.

На вход программе подается строка указанного вида, состоящая не более чем из 9 цифр, разделенных символами умножения '*'. Выведите результат перемножения этих цифр.

Пример входных данных	Пример выходных данных
5*2*3	30

10. Даны два «длинных» целых неотрицательных числа. Требуется найти результат их сложения. В качестве типа данных для хранения чисел используйте строки.

На вход программе подаются два числа, каждое состоит не более чем из 250 цифр и находится в отдельной строке. Выведите результат их сложения.

Пример входных данных	Пример выходных данных
1234 123	1357

Задачи повышенной сложности

1. Дана последовательность символов, имеющая следующий вид: $p_1q_1p_2q_2p_3 \ldots q_{n-1}p_n$, где p_i — цифра, а q_i — знак арифметического действия из набора $\{+, -, *\}$. Вычислите значение выражения, предполагая, что действия выполняются согласно правилам арифметики.

На вход программе подается строка указанного вида, состоящая не более чем из 9 цифр, разделенных символами арифметических операций. Выведите значение арифметического выражения.

Пример входных данных	Пример выходных данных
5-2*3	-1

2. Стало известно, что противник шифрует текст следующим образом. Сначала определяется количество букв в самом длинном слове текста, его длину обозначим k (словом называется непрерывная последовательность английских букв, слова друг от друга отделяются любыми другими символами, длина слова не превышает 20 символов). Затем каждая английская буква заменяется на букву, стоящую в алфавите на k букв ранее (алфавит считается циклическим, то есть перед буквой A стоит буква Z). Другие символы остаются неизменными. Строчные буквы при этом остаются строчными, а заглавные — заглавными. Расшифруйте найденную шифровку.

На вход программе подается текст шифровки, состоящей не более чем из 250 символов. Выведите исходный текст.

Пример входных данных	Пример выходных данных
`Zb Ra Ca Dab Ra.`	`Ce Ud Fd Gde Ud.`

3. У Васи на клавиатуре не работает клавиша «пробел». Поэтому все тексты он теперь набирает слитно. Напишите программу, которая будет разделять набранный Васей текст на слова из данного словаря[1].

На вход программе сначала подается текст, введенный Васей, — строка из не более чем 100 латинских строчных букв. В следующей строке записано количество слов в словаре n — натуральное число, не превосходящее 2000. В следующих n строках записаны слова из словаря — по одному слову в каждой строке, каждое слово длиной не более 20 латинских строчных букв. Слова записаны в алфавитном порядке.

Выведите Васин текст с пробелами между словами (пробел после последнего слова допустим). Если возможно несколько вариантов разбиения строки на слова, выведите любой из них. Гарантируется, что хотя бы один способ разбиения строки на словарные слова существует. Время работы программы на одном тесте — 1 секунда.

Пример входных данных	Пример выходных данных
`whatcanido` `4` `can` `do` `i` `what`	`what can i do`

4. Определите, является ли введенная строка символов записью оператора присваивания по синтаксису языка Pascal для переменной типа `integer`. Функции и арифметические операции, в том числе унарные, в правой части оператора присваивания не используются (то есть производится присваивание константы или переменной).

На вход программе подается строка, состоящая не более чем из 255 символов. В строке могут встречаться незначащие (лишние) пробелы. Выдайте `YES` или `NO` в зависимости от ответа на вопрос задачи.

Примеры входных данных	Примеры выходных данных
`Lsd := 5`	`YES`
`Asd`	`NO`

[1] Задача московской олимпиады по информатике для 7—9 классов, 2007 год.

Урок 12

Вычислительная сложность алгоритма

Решая различные задачи, вы уже убедились, что практически для всех задач существует несколько различных алгоритмов решения. Какой алгоритм лучше подходит для решения конкретной задачи? По каким критериям следует выбирать алгоритм из множества возможных?

Как правило, мы высказываем суждение об алгоритме на основе его оценки человеком. Мы можем назвать алгоритм сложным и запутанным из-за того, что он обладает разветвленной логической структурой, содержащей много проверок условий и переходов. Однако для компьютера выполнение программы, реализующей такой алгоритм, не составит труда, так как он выполняет одну команду за другой, и для компьютера не важно — операция ли это умножения или проверка условия.

Более того, мы можем написать громоздкий алгоритм, в котором выписаны подряд повторяющиеся действия (без использования циклической структуры). Однако с точки зрения компьютерной реализации практически нет никакой разницы, использован ли в программе оператор цикла (например, 10 раз на экран выводится слово «Привет») или 10 раз последовательно выписаны операторы вывода на экран слова «Привет». Поэтому для оценки эффективности алгоритмов используется понятие *сложности алгоритма*.

Вычислительным процессом, порожденным алгоритмом, называется последовательность шагов алгоритма, пройденных при исполнении этого алгоритма. *Сложность алгоритма* — количество элементарных шагов в вычислительном процессе этого алгоритма как функция от исходных данных. Обратите внимание: именно в вычислительном процессе, а не в самом алгоритме. Очевидно, для сравнения сложности разных алгоритмов необходимо, чтобы сложность подсчитывалась в одних и тех же элементарных действиях.

Временна́я сложность алгоритма — это время T, необходимое для его выполнения в зависимости от исходных данных. Оно равно произведению числа элементарных действий k на среднее время выполнения одного действия t: $T = kt$.

Поскольку t зависит от компьютера, естественно считать, что сложность алгоритма в первую очередь зависит от k. Очевидно, что в наибольшей степени количество операций при выполнении алгоритма определяется количеством обрабатываемых данных. Действительно, для упорядочивания по алфавиту списка из 100 фамилий требуется существенно меньше операций, чем для упорядочивания списка из 100 000 фамилий. Поэтому сложность алгоритма выражают в виде функции от объема входных данных.

Ниже мы рассмотрим, как сложность алгоритмов вычисляется для алгоритмов поиска и сортировки.

Алгоритмы поиска

Задача поиска информации в настоящее время является одной из основных задач, решаемых с помощью компьютера. И не всегда используемые при этом алгоритмы просты и понятны. Мы начнем с, казалось бы, тривиальной задачи поиска элемента в неупорядоченном массиве. Во всех примерах, относящихся к поиску в одномерном массиве, будем использовать следующую переменную a:

```
a: array [0..n] of <тип элемента>;
```

при этом собственно элементы массива, которые мы будем рассматривать, пронумерованы от 1 до n, а нулевой элемент будем использовать в случае необходимости как вспомогательный. Конкретный же тип элемента не важен, он может быть как любым числовым, так и символьным или даже строковым.

Алгоритм поиска в массиве элемента, значение которого равно k, может выглядеть так:

```
i := 0;
repeat
  i := i + 1
until (i = n) or (a[i] = k);
if a[i] = k then write(i) else write(0);
```

При отсутствии в массиве элемента с искомым значением k печатается значение нулевого индекса. Оказывается, и такую программу можно упростить с использованием так называемого «барьерного» метода, который часто применяется в программировании. В данном случае он заключается в том, что мы можем занести в дополнительный элемент массива (например, нулевой) искомое значение k, избавив тем самым условие окончания цикла от проверки на выход за границу массива:

```
a[0] := k;
```

```
i := n;
while (a[i] <> k) do
  i := i - 1;
write(i);
```

Эта программа не просто проще и эффективней. В ней практически невозможно сделать ошибку. Несложно подсчитать количество операций сравнения, которое в худшем случае будет выполнено при работе данного алгоритма. Оно равно $n+1$. Говорят, что данный алгоритм линеен от входных данных.

Рассмотрим теперь массив, элементы которого упорядочены по неубыванию, то есть $a_1 \leqslant a_2 \leqslant \ldots \leqslant a_n$.

Поиск элемента, равного k, в таком массиве осуществляется с помощью алгоритма деления пополам. Приведем пример эффективной реализации данного алгоритма. В ней предполагается, что если искомый элемент находится в массиве, то он расположен между элементами с индексами $L+1$ и R включительно:

```
L := 0; R := n;
while R - L > 1 do
begin
  m := (L + R) div 2;
  if a[m] < k then L := m
              else R := m
end;
if a[R] = k then write(R) else write(0)
```

На каждом шаге этого алгоритма длина части массива, в которой мы осуществляем поиск элемента, уменьшается в два раза, то есть если $n=2^i$, то алгоритм будет выполнять $i+1$ сравнений. Эта величина существенно меньше, чем n. Так, для $n=1000$ число сравнений будет равно 11, а для $n=10^6$ — всего 21.

Алгоритмы сортировки

Задача сортировки массива, то есть перестановки элементов массива так, чтобы они были упорядочены по возрастанию, убыванию или другой аналогичной характеристике, является одной из основных технических задач программирования. С этой задачей мы сталкиваемся и при записи фамилий учеников в классном журнале, и при подведении итогов соревнований, и даже при упорядочении игральных карт, например при игре в преферанс. Рассмотрим простые алгоритмы сортировки массивов и подсчитаем их вычислительную сложность. Научитесь быстро реализовывать один из алгоритмов сорти-

ровки, чтобы в дальнейшем в случае необходимости не тратить время на его отладку, так как очень часто сортировка используется как первый шаг в алгоритме решения более сложной задачи.

Традиционно наиболее простой в реализации считается так называемая «пузырьковая» сортировка. Суть ее в случае упорядочения по неубыванию заключается в следующем. Будем просматривать слева направо все пары соседних элементов: a_1 и a_2, a_2 и a_3, ..., a_{n-1} и a_n. Если при этом $a_i > a_{i+1}$, то элементы меняем местами. В результате такого просмотра массива максимальный элемент окажется на крайнем справа (своем) месте. Об остальных элементах ничего определенного сказать нельзя. Будем просматривать массив снова, исключив из рассмотрения правый элемент. На своем месте теперь окажется уже второй по величине элемент. И так далее. В последнем просмотре будут участвовать только первый и второй элементы. Общее число просмотров при этом равно $n-1$. Приведем фрагмент программы, реализующий описанный алгоритм:

```
for j := 1 to n - 1 do {цикл по просмотрам}
  for i := 1 to n - j do {просмотр массива}
     if a[i] > a[i + 1] then
     begin
       x := a[i];
       a[i] := a[i + 1];
       a[i + 1] := x
     end;
```

При оценке вычислительной сложности алгоритмов сортировки обычно отдельно подсчитывают количество операций сравнения и количество операций присваивания, так как заранее неизвестно, какая из них окажется более трудоемкой, ведь сравниваться между собой могут не только числа, но и строки, и другие достаточно сложные объекты. Иногда обе операции сопоставимы по трудоемкости.

Количество сравнений в данном алгоритме равно

$$(n-1)+(n-2)+(n-3)+\ldots+1=\frac{n(n-1)}{2}.$$

Количество присваиваний в три раза больше, чем число выполненных обменов. В худшем случае (когда изначально массив упорядочен по убыванию) обмен будет производиться после каждого сравнения и общее число присваиваний будет равно $\frac{3n(n-1)}{2}$.

Говорят, что данный алгоритм квадратичный как по числу сравнений, так и по числу присваиваний. «Пузырьковым» он называется потому, что в результате каждого просмотра максимальный из

оставшихся элементов оказывается на своем месте — «всплывает». По-другому такая сортировка называется обменной.

Рассмотрим алгоритм сортировки, основанный на принципиально иной идее. Найдем минимальный элемент в массиве и поменяем его с первым элементом массива. В результате он окажется на своем месте. Затем найдем минимальный элемент среди оставшихся и поменяем его со вторым элементом. На $(n-1)$-м шаге мы закончим упорядочивание нашего массива. Такой алгоритм называется сортировкой прямым выбором. Приведем фрагмент программы, реализующий описанный алгоритм:

```
for i := 1 to n - 1 do
begin
  mini := i;
  for j := i + 1 to n do
    {ищем номер минимального элемента}
    if a[j] < a[mini] then mini := j
  {меняем минимальный элемент с i-м}
  x := a[i];
  a[i] := a[mini];
  a[mini] := x
end;
```

Количество выполняемых операций в данном алгоритме всегда одинаково. Для сравнений оно равно

$$(n-1)+(n-2)+(n-3)+\ldots+1=\frac{n(n-1)}{2},$$

а для присваиваний — всего $3(n-1)$.

Итак, данный алгоритм квадратичный по числу сравнений и линейный (эффективный) по числу присваиваний.

Пусть теперь нам надо отсортировать массив, состоящий из десятичных цифр. Оказывается, в этом случае существует гораздо более эффективный алгоритм сортировки по сравнению с рассмотренными выше. А именно, подсчитаем во вспомогательном массиве `d` количество вхождения каждой из цифр в исходный массив `a`. Сделать это можно за один проход массива `a`. А затем заполним массив `a` заново согласно значениям массива `d`:

```
const maxn = 10000;
var d: array[0..9] of integer; {массив для подсчета}
    a: array[1..maxn] of 0..9; {массив цифр}
    n, i, j, k: integer;
begin
```

```
  readln(n); {n - количество цифр}
  for i := 1 to n do a[i] := random(10);
  for j := 0 to 9 do d[j] := 0;
  for i := 1 to n do d[a[i]] := d[a[i]] + 1;
  k := 0;
  for j := 0 to 9 do {заполняем a заново}
    for i := 1 to d[j] do
    begin
      k := k + 1;
      a[k] := j
    end
end.
```

Заметим, что эта программа работает верно и в случае, когда какой-либо из цифр во вводимой последовательности нет совсем. Подобный алгоритм называют сортировкой подсчетом. Его вычислительная сложность в общем случае составляет $2n+2m$ операций, где m — количество различных значений среди элементов в массиве (в нашем примере их было всего 10). Очевидно, что если $m \leqslant n$ и мы можем отвести память для подсчета количества каждого из m возможных значений для элементов массива, то алгоритм окажется линейным относительно n.

Задачи

1. Словарь задан массивом отсортированных в лексикографическом порядке строк. Напишите программу эффективного поиска слова в словаре.

На вход программе сначала подается искомое слово, во второй строке — число n ($1 \leqslant n \leqslant 100\,000$) — количество слов в словаре. В следующих n строках расположены слова словаря, по одному слову в строке. Все слова состоят только из строчных латинских букв, слова упорядочены по алфавиту (расположены в лексикографическом порядке). Длина слов не превосходит 20. Пустых слов нет. Выведите `yes` или `no` в зависимости от того, есть искомое слово в словаре или нет.

Пример входных данных	Пример выходных данных
abba 4 a ab aba baba	no

2. Напишите эффективную программу поиска числа k в двумерном массиве размера $n \times m$, элементы которого в каждой строке и столбце упорядочены по неубыванию.

На вход программе сначала подаются числа n и m ($1 \leqslant n, m \leqslant 5000$). Далее следуют n строк по m чисел в каждой, состоящих из чисел, не превосходящих 30 000, упорядоченных так, как указано в условии задачи. В последней строке находится искомое число k. Выведите `yes` или `no` в зависимости от того, есть искомое число в массиве или нет. Число действий в алгоритме поиска должно быть порядка $n + m$.

Пример входных данных	Пример выходных данных
`3 4` `1 2 3 4` `2 3 4 5` `3 4 5 6` `5`	`yes`

3. Измените алгоритм «пузырьковой» сортировки так, чтобы он заканчивал свою работу в случае, когда на очередном проходе не произошло ни одного обмена (это означает, что массив уже отсортирован и дальнейшие проходы не нужны).

На вход программе сначала подается значение $n \leqslant 10\,000$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30 000. Распечатайте отсортированный по неубыванию массив.

Пример входных данных	Пример выходных данных
`5` `3 4 2 5 1`	`1 2 3 4 5`

4. Реализуйте алгоритм сортировки подсчетом для произвольных чисел по модулю не превосходящих 10 000.

На вход программе сначала подается значение $n \leqslant 100\,000$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 10 000. Распечатайте отсортированный по неубыванию массив.

Пример входных данных	Пример выходных данных
`5` `3 4 2 5 1`	`1 2 3 4 5`

5. Реализуйте алгоритм сортировки вставками, который заключается в том, что мы просматриваем последовательно a_2, a_3, a_4, …, a_n и каждый новый элемент a_i вставляем на подходящее место в уже упорядоченную совокупность a_1, a_2, …, a_{i-1}. Это место определяется последовательным сравнением a_i с упорядоченными элементами a_{i-1}, …, a_1 и обменом с теми элементами, которые больше, чем a_i. Такой алгоритм называется сортировкой вставками. Именно его обычно использует человек в быту при упорядочении предметов, например библиографических карточек в каталоге библиотеки.

На вход программе сначала подается значение $n \leqslant 10\,000$ — количество элементов в массиве. В следующей строке входных данных расположены сами элементы массива — целые числа, по модулю не превосходящие 30 000. Распечатайте отсортированный по неубыванию массив.

Пример входных данных	Пример выходных данных
5 3 4 2 5 1	1 2 3 4 5

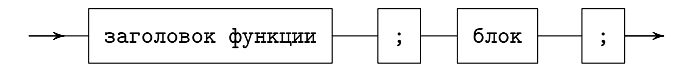
Урок 13

Подпрограммы

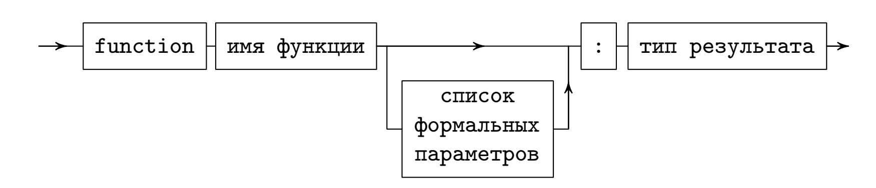
Функции

Ранее мы познакомились с различными стандартными функциями языка Pascal. Кроме них программист может воспользоваться своими собственными функциями, предварительно описав их. Описание функции располагается до тела основной программы, то есть оно входит в раздел описаний. Этим описанием задается часть программы, в которой по набору входных параметров вычисляется и возвращается некоторое значение, не обязательно числовое.

Описание функции:



В *заголовке функции* указываются имя функции, формальные параметры (если они имеются) и тип результата функции:



Результат функции может быть любого скалярного типа, а также типа `string`. В Delphi синтаксис функций расширен, и результатом функции может быть и запись (`record`), и массив. Формальное описание блока (в данном случае его еще называют телом функции) можно найти во введении. В операторной части блока функции задаются операторы, которые будут выполняться при вызове (активизации) функции. Среди них должен содержаться по крайней мере один оператор присваивания, в котором имени функции присваивается значение, тип которого совпадает с типом результата функции. Результатом функции будет последнее присвоенное значение. Если такой оператор присваивания отсутствует или он не был выполнен, то значение, возвращаемое функцией, не определено.

В языке Delphi существует и другой, более удобный способ обозначения результата функции — это стандартная переменная с именем `result`. Эта переменная того типа, который указан в качестве типа результата. При использовании в описании функции такой предопределенной переменной результатом функции является последнее значение, присвоенное этой переменной.

После описания функции ее можно использовать в выражениях наряду со стандартными функциями. Функция активируется только при вызове ее из тела программы или из другой вызванной подпрограммы (процедуры или функции). В последнем случае вызываемая функция (или ее заголовок) должна быть описана выше.

При вызове функции указываются ее имя и фактические параметры, необходимые для вычисления функции. Отвечающие друг другу формальные и фактические параметры должны быть одного и того же типа. О видах параметров и механизме их передачи будет рассказано ниже. Во время вычисления выражения функция, входящая в него, выполняется и значением соответствующего операнда становится результат, возвращаемый функцией.

Если имя функции используется при вызове функции внутри ее же описания, то функция выполняется рекурсивно. Для переменной `result`, используемой в выражениях вместо имени функции, это не так. Подробнее с рекурсией мы познакомимся на следующем уроке.

Пример 1. *Приведем пример программы нахождения максимума из трех чисел, использующей функцию*:

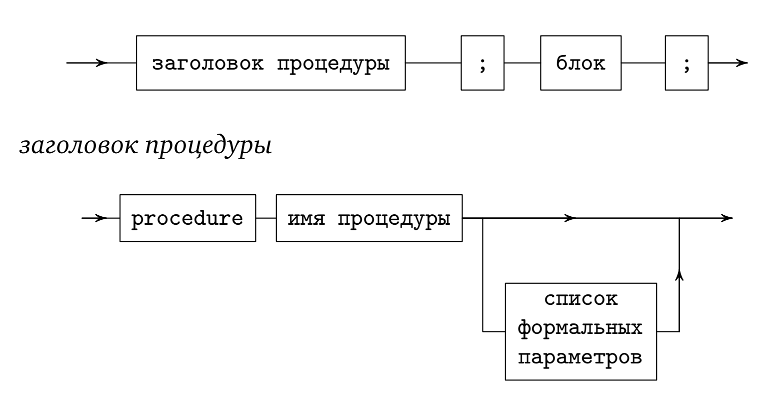
```
var x, y, z: integer;
function max(a, b: integer): integer;
begin
  if a > b then max := a
           else max := b
end;
begin
  readln(x, y, z);
  writeln(max(max(x, y), z))
end.
```

Процедуры

Другим видом подпрограмм в языке Pascal являются процедуры. По мере прогресса в искусстве программирования, как пишет автор языка Pascal Н. Вирт, программы стали создаваться методом последовательных уточнений. На каждом этапе программист разбивает задачу на некоторое число подзадач. Концепция *процедур* (то есть под-

программ) позволяет выделить подзадачу как отдельную подпрограмму. Описание процедуры служит для сопоставления части программы с некоторым именем. Описание выглядит как программа, вместо заголовка которой фигурирует заголовок процедуры.

описание процедуры



Процедура не обязана возвращать значение, поэтому в ее заголовке нет такого элемента, как тип результата. Ее имя в теле процедуры также нельзя использовать для присваивания каких-либо значений. Тело процедуры, как и функции, представляет собой программный блок.

Процедура активируется с помощью оператора процедуры. Он представляет собой то же имя, что и в заголовке процедуры, с перечислением в скобках фактических параметров (если процедура параметров не содержит, то скобки опускаются, как, например, при вызове процедуры `readln` без параметров).

Пример 2. *Приведем пример описания и вызова процедуры, которая печатает первые n элементов массива*:

```
type aa = array[1..100] of integer;
var a: aa;
    i: integer;
procedure print(n: integer; var m: aa);
var i: integer;
begin
  for i := 1 to n do
    write(m[i], ' ');
  writeln
end;
```

```
begin
  for i := 1 to n do a[i] := random(100);
  for i := 1 to n do print(i, a)
end.
```

Не запуская данную программу, определите, как будет выглядеть результат ее работы на экране.

Обратите внимание на то, что параметр цикла `for`, используемого внутри процедуры в примере 2, описан внутри соответствующей процедуры. В языке Delphi это стало фактически правилом: параметр цикла должен быть описан внутри того блока, в котором он используется. Связано это с аппаратной реализацией цикла `for` и различными областями памяти, отводимыми под глобальные и локальные переменные. Расскажем об этом подробнее.

Параметры процедур и функций

В описании процедуры или функции задается список формальных параметров. Каждый параметр, описанный в этом списке, является локальным по отношению к описываемой процедуре или функции, то есть на него можно ссылаться по его имени из данной подпрограммы, но не из основной программы (использовать в подпрограмме, но не в программе). В общем случае можно сказать, что областью действия переменной является блок, в котором она описана. Так, глобальные переменные можно использовать в любом месте программы, если соответствующие имена не используются еще и в качестве локальных переменных.

Существуют два основных типа параметров: параметры-значения и параметры-переменные. При описании они обозначаются следующим образом:

1) группа параметров, перед которыми отсутствует ключевое слово `var`, является списком параметров-значений;

2) группа параметров, перед которыми стоит ключевое слово `var`, является списком параметров-переменных.

Формальный параметр-значение обрабатывается как локальная по отношению к процедуре или функции переменная, а свое начальное значение он получает из соответствующего фактического параметра при активизации процедуры или функции. Изменения, которые претерпевает формальный параметр-значение, не влияют на значение фактического параметра. Соответствующее фактическое значение параметра-значения должно быть выражением типа, совме-

стимого по присваиванию с типом формального параметра-значения (в том числе оно может быть константой или переменной).

Параметр-переменная используется, когда значение должно возвращаться из процедуры или функции в вызывающем блоке. Соответствующий фактический параметр в операторе вызова процедуры или функции должен быть именем переменной и не может быть выражением, причем переменная должна быть того же типа, что и формальный параметр. При вызове процедуры или функции формальный параметр-переменная замещается ссылкой на фактическую переменную, то есть любое обращение к формальному параметру-переменной внутри подпрограммы приводит к доступу к самому фактическому параметру. Соответственно любые изменения в значении формального параметра-переменной непосредственно производятся с фактическим параметром, причем в процессе выполнения подпрограммы, а не возвращаются в момент ее окончания.

Локальные параметры-значения, ссылки на параметры-переменные и все локальные переменные, описанные непосредственно в подпрограмме, располагаются в специальной области оперативной памяти, называемой *стеком*. Размер стека определяется заранее, например с помощью директив компилятора. В Borland Pascal он не может превышать 65 520 байт, в Delphi он ограничен лишь возможностями используемого компьютера и операционной системы. В системе LINUX максимальный размер стека, выделяемого одной программе, устанавливается в операционной системе, а установки самой программы игнорируются.

Пример 3. *Рассмотрим механизм передачи параметров на примере.*

```
var a, b: integer;
procedure ab(a: integer; var b: integer);
begin
  a := a + 1;
  b := b + 1;
  writeln(a,' ',b)
end;
begin
  a := 1;
  b := 2;
  ab(a + 2, b);
  ab(b, a);
  writeln(a, ' ', b)
end.
```

Попробуйте, не запуская эту программу на выполнение, определить, что она выдаст в качестве результата своей работы, а потом сравните с результатом ее выполнения на компьютере. Давайте разберемся, почему результат получается именно такой. При первом обращении к процедуре `ab` формальный параметр `a` примет значение выражения `a + 2` (оно равно 3), где `a` — это уже глобальная переменная. Второй параметр `b` в данном случае означает, что под переменной `b` в подпрограмме будет подразумеваться глобальная переменная `b`. Совпадение имен здесь никакой роли не играет. При выполнении процедуры локальная переменная `a` станет равна 4, а локальная переменная `b`, она же глобальная переменная `b`, станет равна 3. Оператор вывода при этом напечатает:

```
4 3
```

При следующем обращении к процедуре `ab` локальная переменная `a` примет значение глобальной переменной `b`, а под формальным параметром `b` теперь будет подразумеваться глобальная переменная `a`. Тогда вывод процедуры будет таким:

```
4 2
```

При этом значение глобальной переменной `b` останется неизменным, глобальная же переменная `a` теперь равна 2. Поэтому последний оператор в программе напечатает следующее:

```
2 3
```

Попробуем сформулировать правила, по которым следует определять вид того или иного параметра процедуры или функции:

1) входные параметры скалярных и строковых типов, значения которых не должны измениться в программе, должны быть параметрами-значениями, в этом случае при обращении к соответствующей процедуре или функции можно использовать не только имена переменных, но и константы, и выражения; в большинстве стандартных функций, таких, например, как `sin(x)`, параметры описаны именно как параметры-значения;

2) выходные (результирующие) параметры должны быть параметрами-переменными;

3) любые параметры составных типов, таких, например, как массивы, практически всегда должны быть параметрами-переменными; в противном случае в стеке будет заведен новый массив, размер которого совпадает с исходным, а значения будут скопированы из исходного, так использовать компьютерную память нецелесообразно;

4) переменные файловых типов (см. урок 15) могут передаваться только как параметры-переменные.

Задачи

1. Напишите функцию для нахождения наибольшего общего делителя двух чисел с помощью алгоритма Евклида и используйте ее в программе для нахождения НОД уже n чисел.

На вход программе сначала подается значение n $(2 \leqslant n \leqslant 100)$. В следующей строке находятся n целых неотрицательных чисел, не превосходящих 30 000. Выдайте их НОД.

Примеры входных данных	Примеры выходных данных
`3` `24 8 20`	`4`
`4` `0 2 4 8`	`2`

2. Напишите функцию, вычисляющую длину отрезка по координатам его концов. С помощью этой функции напишите программу, вычисляющую периметр треугольника по координатам трех его вершин.

На вход программе подается 6 целых чисел — координат x_1, y_1, x_2, y_2, x_3, y_3 вершин треугольника. Все числа по модулю не превосходят 30 000. Выдайте значение периметра этого треугольника с точностью до 6 знаков после десятичной точки.

Пример входных данных	Пример выходных данных
`0 0 1 0 0 1`	`3.414214`

3. Напишите функцию, вычисляющую площадь треугольника по целочисленным координатам трех его вершин. Стандартные вещественные функции, такие как sin, cos, sqrt и другие, не использовать.

На вход программе подается 6 целых чисел — координат x_1, y_1, x_2, y_2, x_3, y_3 вершин треугольника. Все числа по модулю не превосходят 30 000. Выдайте значение площади этого треугольника с точностью до двух знаков после десятичной точки.

Пример входных данных	Пример выходных данных
`0 0 1 0 0 1`	`0.50`

4. Напишите процедуру печати двумерного массива. В программе задайте массив случайным образом, распечатайте его с использованием процедуры, затем поверните данный массив на 180 градусов и распечатайте его еще раз.

5. Напишите процедуру, которая по входному параметру n — нечетному числу, не превосходящему 21, будет печатать с помощью символов «*» равнобедренный треугольник следующего вида:

```
  *
 ***
*****
```

Используя эту процедуру, нарисуйте на экране елочку. Для этого введите в процедуру дополнительный параметр: количество пробелов, на которое должен отстоять треугольник от левого края экрана.

6. Напишите процедуры печати одномерного массива и его сортировки одним из алгоритмов. В программе задайте массив случайным образом, распечатайте его с использованием процедуры, затем отсортируйте и распечатайте массив еще раз.

7. Напишите процедуру `swap` обмена местами значений двух целочисленных переменных без использования вспомогательной переменной. Объясните, почему данная процедура работает некорректно при обращении к ней с одной и той же переменной, например `swap(a,a)`.

8. Напишите функцию, вычисляющую значение интеграла функции $f(x)$ на отрезке $[a;b]$ (или площадь под графиком функции) одним из численных методов (например, методом трапеций, прямоугольников или методом «Монте-Карло»). Параметрами такой функции должны быть сама функция f (функцию тоже можно передавать как параметр), вещественные значения a и b, а также точность вычислений ε. Результатом является значение интеграла.

9. Напишите функцию, решающую уравнение $f(x)=0$ на отрезке $[a;b]$ в предположении, что на этом интервале функция непрерывна и уравнение содержит ровно один корень. Используйте один из численных методов (например, метод деления пополам или метод хорд). Параметрами такой функции должны быть сама функция f, вещественные значения a и b, а также точность вычисления корня ε. Результатом является искомый корень.

10. Напишите процедуру, рисующую график непрерывной функции $f(x)$ на отрезке $[a;\ b]$. Параметрами такой процедуры должны быть сама функция f, вещественные значения a и b, а также координаты левого верхнего и правого нижнего углов прямоугольной части экрана (окна), в которой должен быть нарисован график.

Урок 14

Рекурсия

Рекурсия является средством программирования, при котором процедура или функция прямо или косвенно вызывает сама себя. При прямой рекурсии в описании функции или процедуры используется обращение к ней же самой, но с другим набором параметров. При косвенной рекурсии подпрограмма вызывает какую-либо другую подпрограмму, в описании которой содержится вызов исходной подпрограммы (например, процедура `A` вызывает процедуру `B`, а процедура `B` вызывает процедуру `A`). Косвенный вызов может быть организован и более сложным образом, то есть в рекурсию могут быть вовлечены несколько подпрограмм. Покажем, как на языке Pascal можно описать подобную систему подпрограмм (без специальных средств это сделать невозможно, так как при описании процедуры `A` процедура `B` уже должна быть описана и наоборот):

```
procedure Carl(n: integer); forward;
procedure Clara(x, y: real);
begin
  ...
  Carl(5);
  ...
end;
procedure Carl;
begin
  ...
  Clara(8.3, 2.4);
  ...
end;
```

Таким образом, сначала производится так называемое предописание одной из процедур, которое состоит из заголовка с полным списком параметров и слова `forward`. После этого к данной процедуре уже можно обращаться при описании других подпрограмм. При настоящем же ее описании список параметров опускается.

Рекурсия позволяет, например, очень просто запрограммировать вычисление рекуррентных соотношений. Рассмотрим следующий спо-

соб описания вычисления факториала:

$$f_0 = 1, \quad f_n = f_{n-1} \cdot n.$$

Соответствующая данному описанию функция на языке Pascal выглядит так:

```
function f(n: integer): longint;
begin
  if n = 0 then f := 1
           else f := f(n - 1) * n
end;
```

Рекурсивный вариант реализации алгоритма обычно выглядит изящнее и дает более компактный текст программы, но исполняется медленнее. Кроме того, при каждом рекурсивном вызове (в приведенном примере их $n+1$) локальные переменные и параметры рекурсивной подпрограммы размещаются в стеке, то есть в данном случае используется значительно больше памяти, чем при нерекурсивном вычислении того же факториала. Часто такое избыточное использование ресурсов компьютера может оказаться критичным для работы программы и она заканчивает свою работу с сообщением *Stack overflow* (переполнение стека) или зависает.

В книге Н. И. Вьюковой, В. А. Галатенко, А. Б. Ходулева «Систематический подход к программированию» приведена интересная иллюстрация, показывающая разницу между рекурсивным и нерекурсивным алгоритмом. Приведем ее с небольшими изменениями.

Пусть у нас имеются n человек, причем k-й $(0 < k < n)$ знает только телефон следующего $(k+1)$-го человека, а n-й знает только то, что он последний. Пусть, далее, некто, знающий телефон первого человека, решил выяснить, чему равно n. Если он будет действовать нерекурсивно, то он запасется бумагой, на которой будет вести подсчеты, напишет на ней 0 и позвонит первому. Затем он сотрет 0, напишет 1 и спросит у первого телефон второго человека, повесит трубку, наберет номер второго, исправит 1 на 2, спросит телефон третьего и т. д., пока не доберется до человека, который сообщит, что он последний. Число, записанное в этот момент на бумаге, и будет служить ответом.

Если же упомянутый некто захочет действовать рекурсивно, он позвонит первому и прикажет: «Алло, не вешайте трубку! Сообщите мне, сколько вас». Поскольку первый не знает, сколько их, а трубку вешать нельзя, ему придется взять мобильный телефон и позвонить второму на домашний, сказав те же слова и оставшись ждать ответа. Второй также возьмет мобильный и т. д., пока, наконец, последний не сообщит предпоследнему, что он один. Предпоследний приба-

вит к ответу последнего 1 и скажет, что их двое, и т. д. Наконец, заждавшийся первый услышит в трубке мобильного голос второго, прибавит к услышанному числу 1 и пойдет к домашнему телефону, чтобы сообщить результат.

Таким образом, при нерекурсивном способе в каждый момент соединена только одна пара абонентов и используется один листок бумаги, при рекурсивном — до n пар и n листков бумаги. Переполнение стека можно сравнить в данном случае с ситуацией, когда АТС окажется перегруженной и к очередному человеку дозвониться не удастся.

Тем не менее рассмотрим примеры уместного использования рекурсивных алгоритмов. В задаче 2 к уроку 5 был приведен пример алгоритма эффективного возведения вещественного числа x в целую неотрицательную степень n. Соответствующая программа не является достаточно прозрачной. Рассмотрим рекурсивный алгоритм решения этой же задачи, основанный на следующих очевидных соотношениях:

$x^0 = 1;$

если n — нечетное, то $x^n = x^{n-1} \cdot n$, в противном случае $x^n = (x^2)^{n/2}$.

Зададим по этому описанию рекурсивную функцию:

```
function power(x: real; n: integer): integer;
begin
  if n = 0 then power := 1
  else if n mod 2 = 0 then
            power := power(x * x, n div 2)
       else power := power(x, n - 1) * x
end;
```

В качестве другой иллюстрации приведем решение известной головоломки «Ханойские башни». В этой задаче имеются три стержня, обозначенных буквами A, B и C. На стержне A, как на детской пирамидке, надеты n колец в порядке убывания диаметров. Требуется переложить все кольца на стержень C, используя стержень B как вспомогательный. За один ход разрешается переложить верхнее кольцо с любого стержня на любой другой, при этом большее кольцо нельзя класть на меньшее.

Будем рассуждать так. Для нуля колец задача уже решена. Пусть мы умеем решать задачу для $n-1$ колец. Тогда сначала переложим их с A на B, используя стержень C как вспомогательный (это можно сделать, так как n-е кольцо больше всех остальных и процессу перекладывания не мешает). Затем переложим n-е кольцо на стержень C и

снова выполним процедуру перекладывания $n-1$ колец, но теперь уже с B на C, используя стержень A как вспомогательный.

Приведем программу для решения исходной задачи, использующую рекурсивную процедуру, соответствующую описанным действиям.

```
var n: integer;
procedure ABC(n: integer; a, b, c: char);
{a-исходный, b-вспомогательный, c-целевой}
begin
  if n > 0 then
    begin
      abc(n - 1, a, c, b);
      writeln(a, ' -> ', c);
      abc(n - 1, b, a, c)
    end
end;
begin
  readln(n);
  abc(n, 'A', 'B', 'C')
end.
```

Так, для трех колец данная программа выдаст следующий план перекладываний:

```
A -> C
A -> B
C -> B
A -> C
B -> A
B -> C
A -> C
```

Таким образом, мы получили достаточно изящное решение головоломки, преобразовать которое в нерекурсивное не так-то просто.

При анализе рекурсивной программы возникают два вопроса:

а) почему программа заканчивает работу;

б) почему она работает правильно?

Для ответа на вопрос б) достаточно проверить, что содержащая рекурсивный вызов подпрограмма работает правильно, предположив, что вызываемая ею одноименная подпрограмма работает правильно. В самом деле, в этом случае в цепочке рекурсивно вызываемых программ все программы работают правильно (убеждаемся в этом, идя от конца цепочки к началу).

Чтобы ответить на вопрос а), обычно проверяют, что с каждым рекурсивным вызовом значение какого-то параметра изменяется (чаще всего уменьшается), и это не может продолжаться бесконечно. При каких-то значениях параметров рекурсивный вызов происходить не должен.

Задачи

1. Запишите рекурсивную функцию, вычисляющую сумму целых чисел m и n, в которой из арифметических операций используется только прибавление и вычитание единицы.

2. Числа Фибоначчи задаются следующими соотношениями:

$$f_0 = f_1 = 1; \quad f_n = f_{n-1} + f_{n-2}, \quad n > 1.$$

Напишите рекурсивную и нерекурсивную функции, вычисляющие n-е число Фибоначчи, и сравните скорость их работы. Попробуйте объяснить результаты сравнения.

3. Напишите рекурсивную подпрограмму, осуществляющую поиск числа k в упорядоченном по неубыванию числовом массиве, используйте метод деления пополам.

На вход программе сначала подается искомое число k, затем количество элементов в массиве $n \leqslant 10\,000$, затем сами n элементов упорядоченного массива целых чисел, по модулю не превосходящих 30 000. Выведите номер этого элемента или 0, если искомый элемент в массиве отсутствует. Нумерация элементов ведется с единицы. Если искомых элементов несколько, то выведите номер первого из них.

Примеры входных данных	Примеры выходных данных
`5 4` `1 2 3 4`	`0`
`3 6` `1 2 2 3 3 3`	`4`

4. Реализуйте следующий рекурсивный алгоритм сортировки: если массив содержит не более одного элемента, то он упорядочен. Иначе преобразовываем его следующим образом: вначале идут элементы, меньшие первого, затем первый элемент, затем элементы, большие первого. Далее алгоритм применяется к первой и второй группам еще не упорядоченных элементов.

На вход программе в первой строке подается количество элементов n ($n \leqslant 10\,000$). Далее следуют n элементов массива — целые числа,

по модулю не превосходящие 30 000. Выдайте этот же массив чисел после сортировки по неубыванию.

Пример входных данных	Пример выходных данных
`5` `3 4 2 1 5`	`1 2 3 4 5`

5. Опишите рекурсивную функцию, которая определяет, является ли симметричной часть строки S, начиная с i-го элемента и заканчивая j-м. Решите с помощью этой функции задачу 3 к уроку 11.

6. Напишите рекурсивную процедуру для перевода десятичного числа в P-ичную систему счисления, не использующую в своей работе массив.

На вход программе сначала подается значение P $(1 < P < 10)$, а во второй строке — десятичное число. Вывод осуществляйте следующим образом: сначала выведите введенное число в десятичной системе счисления, за ним укажите его систему счисления в круглых скобках, то есть (10), затем ставится знак «=» и аналогично выводится результат работы вашей программы — число в P-ичной системе счисления. Весь вывод осуществляется без пробелов.

Пример входных данных	Пример выходных данных
`3` `123`	`123(10)=11120(3)`

7. Запрограммируйте процедуру, которая будет печатать все сочетания из n первых натуральных чисел по k чисел.

На вход программе подаются натуральные числа n $(n \leqslant 17)$ и k $(k \leqslant n)$. Выведите все k-элементные подмножества множества $\{1, 2, 3, \ldots, n\}$. Каждое подмножество выводите в отдельной строке. Числа внутри одного подмножества упорядочивайте по возрастанию.

Пример входных данных	Пример выходных данных
`3 2`	`1 2` `1 3` `2 3`

8. В первой строке входных данных записаны n — число строчек и m — число столбцов в двумерном массиве. В последующих n строчках записано по m чисел, каждое из которых равно либо 0, либо 1. Числа разделены пробелами. Требуется вывести количество объектов, со-

стоящих из единичек. Если две единички являются соседями по вертикали или горизонтали, то они принадлежат одному и тому же объекту.

Пример входных данных	Пример выходных данных
2 3 1 0 1 0 1 1	2

9. Дана шахматная доска размера $n \times n$. Пусть конь стоит на клетке (1, 1). Необходимо найти такую последовательность ходов коня, при которой он побывает на каждой клетке доски ровно по одному разу.

На вход программе подается натуральное число n ($n \leqslant 8$). Если обход невозможен, то выведите в выходной файл 0, если возможен, то 1, а на следующих строчках выведите матрицу $n \times n$, иллюстрирующую порядок обхода. Выравнивать числа по столбцам не обязательно.

Примечание. Скорость работы рекурсивной программы в этой задаче существенно зависит от порядка, в каком будут рассматриваться варианты хода коня из очередной клетки. Одним из удачных порядков является размещение всех восьми вариантов хода «по кругу».

Примеры входных данных	Примеры выходных данных
3	0
5	1 1 14 19 8 25 6 9 2 13 18 15 20 7 24 3 10 5 22 17 12 21 16 11 4 23

10. Дана шахматная доска размера $n \times n$. Найдите количество всевозможных различных расстановок n ферзей на этой доске. При этом никакие два ферзя не должны «бить» друг друга. Две расстановки считаются различными, если существует такое поле, что в первой расстановке на нем есть ферзь, а во второй — нет.

На вход программе подается натуральное число n ($n \leqslant 10$). Выведите количество всевозможных «безопасных» расстановок n ферзей на этой доске.

Пример входных данных	Пример выходных данных
5	10

Урок 15

Файловые переменные

Словом «файл» в программировании называется сразу несколько несовпадающих понятий. Во-первых, это файл операционной системы — «поименованная часть диска». Во-вторых, это абстрактная структура данных с последовательным доступом, и, в-третьих, переменные файловых типов, реализующие эту структуру данных в конкретном языке программирования.

В этом уроке мы подробно рассмотрим лишь один из файловых типов — `text`. Стандартный файловый тип `text` определяет файл, содержащий символы, сгруппированные в строки. Переменная типа `text` называется файловой переменной. Перед использованием файловой переменной она должна быть связана с внешним файлом операционной системы с помощью вызова процедуры `assign`. Во внешних файлах сохраняется записанная в файл информация, или они служат источниками информации, которая считывается из файла.

Когда связь с внешним файлом установлена, для подготовки к операции ввода или вывода файловая переменная должна быть «открыта» либо на чтение, либо на запись. На чтение можно открыть только реально существующий файл. Делается это с помощью процедуры `reset`, а новый файл можно создать и открыть на запись с помощью процедуры `rewrite`. Если же на запись открывается уже существующий файл, то он автоматически становится пустым в момент открытия, то есть его старое содержимое оказывается безвозвратно утерянным.

С большинством правил чтения данных из текстовых файлов и записи в них результатов работы программы вы знакомы с момента написания самых первых программ. Дело в том, что в момент начала выполнения программы всегда автоматически открываются стандартные текстовые файловые переменные `input` и `output`. `input` — это доступный только для чтения файл, связанный с клавиатурой, а `output` — это открытый на запись файл, связанный с дисплеем. И хорошо знакомые вам процедуры `read` и `write` — это процедуры чтения данных из файла и записи данных в файл соответственно. Если имя файловой переменной при этом не указывается, то считается, что чтение происходит из стандартного файла `input`, а запись — в стандартный файл `output`.

Любой файл, как отображение структуры данных, представляет собой линейную последовательность элементов (в случае текстового файла — символов). Доступ к элементам текстового файла организуется последовательно, то есть когда элемент считывается с помощью процедуры `read` или записывается с помощью процедуры `write`, текущая позиция файла перемещается к следующему по порядку элементу файла.

При открытии текстового файла внешний файл интерпретируется особым образом: считается, что он представляет собой последовательность символов, сгруппированных в строки, где каждая строка заканчивается символом конца строки, за которым в Windows следует еще и символ перехода на новую строку.

Когда программа завершает обработку файла, он должен закрываться с помощью процедуры `close`. Только после полного закрытия файла связанный с ним файл операционной системы обновляется. Затем файловая переменная может быть связана с другим внешним файлом.

Пример 1. *Приведем пример работы с текстовыми файлами*:

```
var  n: integer;
  f, g: text;
begin
  assign(f,'input.txt');
  reset(f);
  read(f, n);
  ...
  assign(g, 'output.txt');
  rewrite(g);
  writeln(g, n);
  close(g)
end.
```

Стандартные процедуры и функции, использующиеся для текстовых файлов

Процедура `assign(файловая переменная, строка)` — ставит в соответствие имя файловой переменной имени внешнего файла, которое задается в строковой переменной или строковой константе.

Процедура `reset(файловая переменная)` — открывает существующий файл на чтение.

Процедура `rewrite(файловая переменная)` — открывает файл на запись.

Процедура append(файловая переменная) — открывает существующий (!!!) файл для дописывания (записи) информации после уже имеющейся.

Процедура close(файловая переменная) — закрывает открытый файл.

Функция eof(файловая переменная): boolean — проверяет, достигнут ли при чтении из файла конец файла (**e**nd-**o**f-**f**ile).

Функция eoln(файловая переменная): boolean — проверяет, достигнут ли при чтении из текстового файла конец строки (**e**nd-**o**f-**lin**e).

Функция SeekEof(файловая переменная): boolean — проверяет, достигнут ли при чтении из файла конец файла, пропуская при этом разделители: пробелы, символы табуляции и перевода строки.

Функция SeekEoln(файловая переменная): boolean — проверяет, достигнут ли при чтении из файла конец строки, пропуская при этом пробелы и символы табуляции.

Процедура read — считывает одно или более значений из файла в одну или более переменных.

Процедура readln — выполняет те же действия, что и процедура read, а затем делает пропуск данных до начала следующей строки текстового файла.

Процедура write — записывает в файл одно или более значений.

Процедура writeln — выполняет те же функции, что процедура write, а затем добавляет к файлу метку конца строки (один или два символа в зависимости от операционной системы).

Ввод и вывод данных с использованием текстовых файлов

Рассмотрим полезные приемы программирования ввода данных различных типов. Начнем с описания считывания из текстового файла или консоли (клавиатуры), которая с точки зрения программы также является текстовым файлом, числовых данных. В условии задачи ввод большого количества чисел может быть задан двумя способами. В первом способе сначала предлагается ввести количество чисел, а уж затем сами эти числа. В этом случае при программировании сложности не возникают. Во втором способе количество чисел приходится определять в процессе их считывания.

Пример 2. *Пусть для каждой строки входного файла f требуется найти среднее арифметическое чисел, расположенных в ней. Количество чисел в каждой из строк и количество строк при этом неизвест-*

но. Наиболее простым и правильным будет следующее решение такой задачи:

```
while not seekeof(f) do
begin
  n := 0;
  s := 0;
  while not seekeoln(f) do
  begin
    read(f, a);
    s := s + a;
    n := n + 1
  end;
  readln(f);
  if n > 0 then writeln(s / n:0:2)
           else writeln
end;
```

Заметим, что обычно применяемые в таких случаях функции `eof` и `eoln` заменены на `seekeof` и `seekeoln` соответственно. Только при показанном способе ввода чисел не возникают ошибки в работе подобной программы, связанные с наличием пробелов в конце строк и пустых строк в конце файла, так как для корректного использования функции `eof` требуется, чтобы признак конца файла стоял непосредственно после последнего числа в файле. То же требование относится к признаку конца строки при использовании функции `eoln`. Отметим, что техническую проблему, связанную с обработкой заранее неизвестного количества чисел в строке или в файле в целом, разрешить на языке программирования C несколько сложнее. Там приходится проверять корректность завершения очередного обращения к функции `scanf`.

Наоборот, если во входном файле находится текст, размер которого неизвестен, то поступать следует несколько по-другому. Использование `seekeoln` может привести к ошибке, так как в тексте пробел уже является значимым символом. С другой стороны, служебные символы, обозначающие конец строки в файле и перевод на новую строку в Windows (их коды 13 и 10), не могут считаться частью текста и не должны анализироваться алгоритмом его обработки. Поэтому если известно, что длина каждой строки текстового файла не превосходит 255 символов (при использовании директивы компилятора `{H+}` в языке Delphi это ограничение снимается), то удобнее всего считывание производить с использованием переменной типа `string`:

```
while not eof(f) do
begin
  readln(f, s);
  if s <> '' then ... {обработать строку s}
end;
```

В этом примере использование `readln`, а не `read` является принципиальным. Если же ограничения на количество символов в одной строке нет, то считывание данных на языке Borland Pascal следует производить посимвольно.

Пример 3. *Посимвольное считывание текста из файла*:

```
while not eof(f) do
begin
  n := 0;
  while not eoln(f) do
  begin
    read(f, c);
    ... {запись символа c в массив или его обработка}
    n := n + 1
  end;
  readln(f); {!!!}
  if n > 0 then ... {обработка строки}
           else ... {строка пустая}
end;
```

Именно использование пустого оператора `readln` позволяет и в этом случае автоматически исключить из рассмотрения символы перевода строки.

Последний пример считывания данных относится к случаю смешанной информации, то есть в файле присутствуют как числа, так и символы или последовательности символов. Формат такого файла обычно определен заранее, поэтому считывание можно организовать сразу в переменные соответствующих типов. Наоборот, считывание информации в одну строковую переменную, а затем выделение из нее отдельных элементов и преобразование строкового представления данных в числовое делает программу более громоздкой и зачастую требует отладки.

При решении таких задач удобно (но не обязательно!) использовать переменные типа `record` (запись). Структура записи содержит фиксированное число компонент, называемых полями. В отличие от массивов компоненты не обязательно одного типа, и непосредственно индексировать их нельзя. Область действия имени поля — запись,

в которой оно определяется. Имена полей можно использовать в качестве имен других переменных вне записи. Все имена полей должны быть различными.

Пусть, например, в каждой строке файла записана фамилия человека, затем через пробел его год рождения и, наконец, опять же через пробел — его пол, обозначенный одной буквой. Для хранения информации о человеке создадим следующую запись:

```
var person: record
              s: string;
              n: integer;
              c: char
            end;
```

Описание записи начинается со слова `record` и заканчивается словом `end`.

Пример 4. *Приведем фрагмент программы, считывающий данные описанного формата из файла сразу в переменные соответствующих типов*:

```
while not seekeof(f) do
begin
  read(f, c);{вспомогательный символ}
  person.s := '';
  while c <> ' ' do {формируем строку с фамилией}
  begin
    person.s := person.s + c;
    read(f, c)
  end;
  read(f, person.n); {считываем год рождения}
  readln(f, c, c);   {считываем пол}
  person.c := c;
  ... {обработка считанной информации}
end;
```

При считывании символа, обозначающего пол человека, предварительно следует пропустить пробел, который ему предшествует. Именно поэтому считываются два символа, а не один, и значение первого символа (пробела) теряется при считывании второго (значения пола).

Во время записи результатов работы программы в файл ошибки могут быть связаны в основном с отсутствием разделителей (пробелов или символов перевода строки) между выведенными в файл числами.

Задачи

1. Дан файл `f.txt`, содержащий сведения об игрушках: в каждой строке указывается четырехзначный артикул изделия, название игрушки, ее стоимость в рублях и возрастные границы детей, для которых игрушка предназначена. Например:

```
1234 кукла 137.50 5 12
```

Запросите с клавиатуры возраст ребенка и сумму денег, которую покупатель может потратить на покупку. Запишите в файл `g.txt` следующую информацию: названия и цену игрушек, которые подходят детям указанного возраста, а цена каждой из них не превышает сумму денег, указанную пользователем.

2. Сведения об ученике, записанные в файле `f.txt`, состоят из его фамилии и названия класса (год обучения и буква), а также оценок, полученных за четверть. Число оценок в разных классах может быть разным. Запишите в файл `g.txt` фамилии тех учеников, которые не имеют двоек и троек, а их средний балл больше чем 4,5.

Пример входного файла	Пример выходного файла
`Иванов 8а 3 4 5` `Петров 9б 4 5 5 5` `Сидоров 11в 5 5 5 5 3`	`Петров`

3. Дан файл `f.txt`, содержащий некоторый текст. Требуется подсчитать количество строк со словами (непробельными символами, разделенными пробелами или символами перевода строки), а также количество слов в этом тексте. Результат выдайте в файл `g.txt`.

Пример входного файла	Пример выходного файла
`Это текст` `в` `текстовом файле.`	`3` `5`

4. В файле `f.txt` содержится запись клиентов на стрижку по формату

```
<фамилия> <название стрижки>
```

В файле `g.txt` для каждого из мастеров парикмахерской записано, сколько минут он делает ту или иную стрижку, по формату

```
<фамилия> <название стрижки> <время> <название стрижки> <время> ...
```

В файл r.txt для каждого клиента выдайте время, когда он выйдет из парикмахерской, если она открывается в 8 утра, а клиентов обслуживает первый освободившийся мастер в порядке очереди. Формат выходных данных:

<фамилия> HH:MM

Таким образом, на выдачу часов и минут отводится ровно по два разряда. Подразумевается, что все клиенты будут обслужены в текущие сутки. Пример работы программы:

f.txt	g.txt	r.txt
Иванов бокс Петрова каре Сидоров бокс	Быстрый бокс 10 каре 30 Медленный бокс 30 каре 60	Иванов 08:10 Петрова 09:00 Сидоров 08:20

5. В файле f.txt записаны команды, которые поступали в автоматизированную систему управления лифтом. Команды бывают двух типов: внешние (вызов лифта) и внутренние (нажатие кнопок внутри кабины). Внешние команды имеют формат

<номер этажа><пробел><U или D>,

а внутренние — или <номер этажа>, или U, или D. Буквы U и D обозначают направление поездки вверх или вниз. Внутренняя команда U обозначает поездку на последний этаж, а D — на первый. Каждая команда находится в отдельной строке. Известно, что в здании 28 этажей, а каждая новая команда поступает в устройство управления, только если предыдущая команда уже выполнена. Запишите в файл g.txt последовательность этажей, на которых останавливался лифт.

Пример входного файла	Пример выходного файла
1 U 5 3 D D	1 5 3 1

6. Во входном файле meteo.dat находятся 366 строк, которые содержат информацию о среднесуточной температуре всех дней 2008 года. Формат каждой из строк следующий: сначала записана дата в виде dd.mm (на запись номера дня и номера месяца в числовом формате отводится строго два символа, день от месяца отделен точкой), затем через пробел записано значение температуры — число со

знаком плюс или минус, с точностью до 1 цифры после десятичной точки. Данная информация отсортирована по значению температуры, то есть хронологический порядок нарушен.

Требуется написать программу, выводящую в файл `avr.dat` информацию о месяцах, у которых среднемесячная температура наименее отклоняется от среднегодовой. В первой строке вывести среднегодовую температуру. Найденные значения для каждого из месяцев следует выводить в отдельной строке в следующим виде:

`<номер месяца> <среднемесячная температуры> <отклонение от среднегодовой>.`

Средние значения выводить с точностью до одного знака после десятичной точки.

Массив для хранения всех введенных значений не использовать.

Пример входного файла	Пример выходного файла
31.12 -25.5 03.01 -24.7 ...	4.5 10 5.7 1.2

7. Ученики, недавно изучающие программирование, употребляют слишком много операторов `goto`, что является почти недопустимым для структурированной программы. Помогите преподавателю информатики написать программу, которая будет оценивать степень структурированности отлаженной программы школьника на языке Pascal, для начала просто подсчитывая количество операторов `goto` в ней.

В синтаксически верной программе ключевое слово оператора перехода `goto` может стоять или в начале строки, или после пробела, или после одного из символов — ‘;’ , ‘:’, ‘}’, а после него может стоять или пробел, или перевод строки, или символ ‘{‘.

Напомним, что кроме обозначения действительно оператора перехода слово `goto` может встречаться в тексте программы в строковых константах, заключенных в апосторофы, или в комментариях, причем для простоты будем считать, что комментарий всегда начинается с символа ‘{‘, а заканчивается первым встретившимся после этого символом ‘}’, и это единственный вид комментариев. В таких случаях слово `goto` подсчитывать не нужно. Строчные и прописные буквы в Pascal не различимы.

Во входном файле `goto.in` находится текст программы на языке Pascal без синтаксических ошибок. Размер программы не превосходит

64 Кб. В выходном файле goto.out должно оказаться одно число — количество операторов goto в этой программе.

Пример входного файла	Пример выходного файла
Label 1,2; Var I: byte; Begin 1: I := I + 1; if I > 10 then goto 2 else writeln(' goto '); GoTo 1; 2: if I < 10 then goto 1 {else { goto 2;}} end.	3

8. Дан текстовый файл f.txt. В каждой строке имеется по крайней мере два слова, разделенных как минимум одним пробелом. За счет изменения количества пробелов между словами добейтесь того, чтобы пробелы в начале и в конце каждой строки отсутствовали. Количество пробелов между словами должно отличаться не более чем на один (см. задачу 7 к уроку 11). Количество символов в каждой строке должно равняться количеству символов в самой длинной строке файла. Результат работы запишите в файл g.txt.

Пример входного файла	Пример выходного файла
␣один␣␣␣два␣␣␣три один␣два␣три	один␣␣␣два␣␣␣␣три один␣␣␣два␣␣␣␣три

9. Даны 2 текстовых файла f1.txt и f2.txt. В каждом из файлов находится неотрицательное количество целых чисел, отсортированных по неубыванию. Каждое из чисел по модулю не превосходит 10^6. Числа разделены между собой пробелами и/или символами перевода строки. Получите новый текстовый файл f3.txt слиянием двух исходных в отсортированном виде.

Пример работы программы:

f1.txt	f2.txt	f3.txt
1 2 4	2 3	1 2 2 3 4

10. Даны два файла f1.txt и f2.txt. Файл f1.txt — инвентаризационный файл, содержащий сведения о том, сколько изделий каких видов продукции хранится на складе. Файл f2.txt — вспомогательный файл, содержащий сведения о том, на сколько уменьшилось или

увеличилось количество изделий по некоторым видам продукции. Файл `f2.txt` может содержать несколько сообщений по продукции одного вида или не содержать ни одного такого сообщения о некоторых видах продукции. Обновите файл `f1.txt` на основе `f2.txt`. Запросы, содержащиеся в этом файле, обрабатываются последовательно. Если какого-то товара на базе первоначально не было, то запрос просто игнорируется, даже если такой товар на базу «возвращают». Если же товара меньше, чем просит покупатель, то он получит лишь имеющийся товар и запрос оказывается выполнен частично (при дальнейшем возврате аналогичного товара на склад данный запрос выполняться не будет). Если товар изначально был, а сейчас его нет, то запрос игнорируется. Товара, который после выполнения запросов на складе отсутствует, в обновленном файле также быть не должно.

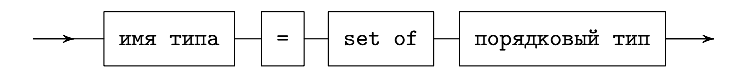
Надо минимизировать количество чтений из файлов (повторных открытий файла на чтение). Массивы в программе использовать нельзя. Пример работы программы:

`f1.txt`	`f2.txt`	измененный `f1.txt`
тушенка 123 сгущенка 321	сгущенка 350 {стало 0} тушенка 140 {стало 0} сгущенка 10 {игнорируется} тушенка -17 {стало 17} тушенка 10 {стало 7} варенье 200 {игнорируется} печенье -10 {игнорируется}	тушенка 7

Урок 16

Тип множество

Описание типа *множество*



Множества в Pascal представляют математические множества без повторений элементов. Множественный тип определяет множество значений, которые представляют собой множество всех подмножеств, составленных из констант базового типа, включая и пустое множество. Если базовый порядковый тип состоит из n элементов, то всего имеется 2^n различных подмножеств множества этих n элементов, и соответствующий множественный тип задает 2^n значений. Количество элементов в базовом типе не может превышать 256. Следовательно, базовыми могут быть типы `byte`, `char`, `boolean`, ограниченный и перечислимый типы.

Константы множественного типа представляют собой перечисление через запятую элементов множества, заключенное в `[]`. Примеры:

```
[], [13], [i + j, i - j], ['A'..'Z', '_', '0'..'9'].
```

Конструкция `[m..n]` означает множество всех элементов i базового типа, для которых выполняется условие $m \leqslant i \leqslant n$. Если $m > n$, то `[m..n]` означает пустое множество `[]`.

Ко всем переменным типа множество применимы следующие операции:

`A + B` — объединение множеств A и B;

`A * B` — пересечение множеств A и B;

`A - B` — множество всех элементов множества A, не являющихся элементами множества B;

Возможны также следующие операции отношения:

`A = B`, `A <> B` — проверка на равенство или неравенство соответственно;

`A <= B`, `A >= B` — проверка на подмножество;

`a in B` — проверка на принадлежность элемента a базового типа множеству B.

Добавить элемент во множество можно с помощью операции объединения +. В этом случае она будет выглядеть так:

```
B := B + [a];
```

Аналогично с помощью операции «–» элемент можно из множества исключить. В обоих случаях элемент обязательно должен быть заключен в квадратные скобки, так как указанные операции производятся над переменными типа «множество». Процедуры `include` и `exclude` ускоряют добавление и удаление отдельных элементов из множества. Связано это с представлением переменных множественных типов в памяти компьютера.

По сути каждая такая переменная представляет собой массив бит, размерность которого равна размерности базового типа (максимальный размер одной переменной 256 бит или 32 байта). Если элемент принадлежит множеству, то на месте соответствующего бита стоит 1, в противном случае — 0. Все операции над множественными переменными являются битовыми операциями, соответственно, они выполняются достаточно быстро и реализованы на аппаратном уровне. Так, операция объединения множеств «+» реализована как побитовая операция «или» (or), а операция пересечения «*» — как побитовое «и».

Как уже говорилось в уроке 8, константы множественного типа удобно использовать в логических выражениях.

Пример 1. *Вместо*

```
if (c='a') or (c='b') or (c='c') or (c='d') or (c='z') then
```

можно писать

```
if c in ['a'..'d', 'z'] then
```

Иногда удобно в программе описать множественную константу, например включающую в себя все символы-разделители языка Borland Pascal:

```
const blanks = [#10, #13, ' ', #09, #26];
```

С помощью такой константы существенно проще, например, подсчитывать количество слов в тексте.

К сожалению, переменные типа множества нельзя ни считывать, ни распечатывать. Поэтому печать элементов множества можно осуществить только с помощью цикла, просматривающего все элементы базового типа и проверяющего каждый из них на принадлежность интересующему нас множеству.

Пример 2. *Нахождение простых чисел до 255 с помощью решета Эратосфена. Найденные простые числа будем помещать во множество `primes`, а в качестве решета использовать множество `sieve`.*

Алгоритм заключается в следующем. Поместим в решето все натуральные числа от двух до 255. Возьмем первое из них (двойку) и уберем из решета числа, кратные двум (каждое второе число). Затем возьмем первое из оставшихся чисел (тройку) и уберем каждое третье число. Далее будем поступать аналогично.

Заметим, что когда мы будем удалять из решета числа, кратные i, то начинать надо с $i \cdot i$, так как числа $2i$, $3i$ и т. д. из решета уже удалены. На каждом шаге минимальное из оставшихся в решете чисел будем помещать во множество простых чисел. Алгоритм можно остановить, когда решето станет пустым. Приведем программу, реализующую данный алгоритм:

```
const n = 255
var sieve, primes: set of 2..n;
   next, j: integer;
begin
  sieve := [2..n];
  primes := [];
  next := 2;
  repeat
    while not (next in sieve) do next := next + 1;
    include(primes, next);
    j := next * next;
    while j <= n do
    begin
      exclude(sieve, j);
      j := j + next
    end
  until sieve = [];
  for j := 2 to 255 do
    if j in sieve then write(j, ' ')
end.
```

Пример 3. *Решим следующую задачу. Станции метро пронумерованы от* 1 *до nmax. Для каждой линии метро известно, какие станции на ней находятся. При этом станция называется пересадочной, если она находится как минимум на двух линиях. Даны номера начальной и конечной станций. Требуется определить, за какое наименьшее число пересадок можно доехать с одной станции до другой*[1].

Решение. Сохраним информацию о линиях метро в массиве множеств *ss*. То есть для каждой линии принадлежащие ей станции будут

[1] Задача московской олимпиады для 7—9 классов 2006 г.

помещены в соответствующее множество. Во множество s будем помещать станции, достижимые из начальной не более чем за k пересадок. Тогда решение задачи будет выглядеть так:

```
const mmax = 100; {количество линий}
      nmax = 200; {количество станций}
type myset = set of 1..200;
var ss: array[1..mmax] of myset;
  a, b: myset;
begin
  ... {считывание данных}
  s := [a]; s1 := [];
  k := 0;
  while not (b in s) and (s1 <> s) do
  begin
    s1 := s; s := [];
    for i := 1 to n do
      if s1 * ss[i] <> 0 then s := s + ss[i]
    k := k + 1;
  end;
  if b in s then writeln(k - 1) else writeln(-1)
end.
```

Если система метро окажется не связанной, и с начальной станции до конечной, не выходя из метро, добраться невозможно, то данная программа также сумеет это определить: если на каком-то шаге ни одной новой станции к достижимым не добавилось, то конечная станция не достижима.

Задачи

1. Двумя способами реализуйте бинарную операцию над множествами — симметрическую разность. Ее результатом являются те элементы двух множеств, которые принадлежат ровно одному из множеств (но не принадлежат их пересечению).

2. Пусть T — множество учителей, уже занятых в определенный час, R — множество занятых кабинетов. Если некоторый предмет ведут все учителя из множества T_1 и проходить он может в любом из кабинетов множества R_1, то требуется определить, может ли этот урок проходить в тот же час, и если может, то изменить множество занятых учителей T и выделить этому предмету один из свободных кабинетов, то есть изменить множество занятых кабинетов R.

3. Дополните решение задачи из примера 3 считыванием данных по следующему формату. На вход программе сначала подается число

n — количество станций метро в городе ($2 \leqslant n \leqslant 200$). Далее вводится число m — количество линий метро ($1 \leqslant m \leqslant 100$). Далее идет описание m линий. Описание каждой линии состоит из числа p_i — количества станций на этой линии ($2 \leqslant p_i \leqslant 50$) и p_i чисел, задающих номера станций, через которые проходит линия (ни через какую станцию линия не проходит дважды).

В конце вводятся два различных числа: A — номер начальной станции и B — номер станции, на которую нам нужно попасть. При этом если через станцию A проходит несколько линий, то мы можем спуститься на любую из них. Также если через станцию B проходит несколько линий, то нам неважно, по какой линии мы на нее приедем.

Выведите минимальное количество пересадок, которое нам понадобится. Если добраться со станции A на станцию B невозможно, выведите одно число -1 (минус один).

Примеры входных данных	Примеры выходных данных
5 2 4 1 2 3 4 2 5 3 3 1	0
5 5 2 1 2 2 1 3 2 2 3 2 3 4 2 4 5 1 5	2
10 2 6 1 3 5 7 4 9 6 2 4 6 8 10 7 3 8	1
4 2 1 2 2 3 4 1 3	-1

4. Напишите программу, которая для двух последовательностей, состоящих из натуральных чисел, не превосходящих n, будет определять, какие числа встречаются в каждой из последовательностей, а какие из чисел от 1 до n — ни в одной из них.

Сначала на вход программе подается число n $(1 \leqslant n \leqslant 255)$. Во второй строке входных данных находятся элементы первой последовательности, в третьей строке — элементы второй последовательности. Выведите в первой строке в порядке возрастания без повторений числа, которые встречаются в каждой из последовательностей, а во второй строке — в порядке возрастания числа от 1 до 255, которые не встречаются ни в одной из них.

Пример входных данных	Пример выходных данных
7 3 2 4 5 2 2 7 4 3 4 2	2 3 4 1 6

5. Напишите программу генерации перестановок n чисел, использующую рекурсивную процедуру и множество для хранения элементов, уже включенных в перестановку.

На вход программе подается число n $(1 \leqslant n \leqslant 12)$. Выведите в лексикографическом порядке все перестановки первых n натуральных чисел, располагая каждую перестановку в отдельной строке.

Пример входных данных	Пример выходных данных
3	1 2 3 1 3 2 2 1 3 2 3 1 3 1 2 3 2 1

Указания и решения

В этом разделе приведены указания и решения наиболее важных или сложных задач курса. Хочется надеяться, что учащиеся будут прибегать к его помощи лишь после того, как ту или иную задачу действительно оказалось сложно решить. Тем не менее, он может оказаться полезен и тогда, когда задача решена и сдана системе автоматической проверки. Возможно, приведенное здесь решение будет более коротким, изящным или эффективным.

Урок 3

Задачи

3. Приведем основной фрагмент решения задачи. Заметим, что вместо операции деления на 0,5, естественным образом возникающей в ходе решения, следует использовать умножение на 2:

```
h := k div 30;
m := k mod 30 * 2;
writeln('It is ', h, ' hours ', m, ' minutes.');
```

6. Данная задача очень важна для различных операций пересчета, связанных с таблицами, нумерация строк и столбцов в которых ведется с единицы (нумеровать с нуля проще!). Формулы, которые кажутся очевидными, обычно оказываются неверными для одного из возможных случаев (обычно для первой или последней строки или для первой страницы), поэтому система тестов для этой задачи должна быть особенно продуманной. Приведем правильное решение:

```
p := (n - 1) div k + 1; {это номер страницы}
l := (n - 1) mod k + 1; {это номер строки}
```

Покажем на примере второй из формул конструктивный способ их получения. Пусть $k = 3$. Впишем в таблицу (см. таблицу на с. 139) остатки от деления на 3 первых семи чисел.

Остатки от деления на 3 лежат в диапазоне `0..2`, а искомый результат — `1..3`. Значит, прибавление 1 к остатку переведет числа в требуемый диапазон. Но тогда каждые 3 последовательных результата оказываются циклически сдвинутыми на единицу относительно правильного ответа. Поэтому остаток от деления на 3 надо брать не от числа n, а от числа $n - 1$, что и приводит нас к желаемому резуль-

n	n mod 3	l	n mod 3 + 1	(n - 1) mod 3 + 1
1	1	**1**	2	**1**
2	2	**2**	3	**2**
3	0	**3**	1	**3**
4	1	**1**	2	**1**
5	2	**2**	3	**2**
6	0	**3**	1	**3**
7	1	**1**	2	**1**

тату. Составление подобных таблиц полезно и при решении других задач.

8. Сложность этой задачи заключается в том, что наиболее естественный способ ее решения — перевести исходную сумму в копейки, умножить ее на n и выделить число рублей и копеек, в данном случае не подходит. При максимальных ограничениях мы получаем, что число $30\,000 \times 100 \times 30\,000 = 90\,000\,000\,000$ не умещается в стандартном типе данных `integer` языка Delphi (или типе `longint` в Borland Pascal). Правильное решение работает непосредственно с рублями и копейками.

9. С помощью этой задачи можно понять, как правильно округлять вещественные числа. Так, следующий способ выделения целого числа копеек из вещественной цены товара r на специально подобранных тестах может оказаться неверным:

```
k := trunc(frac(r) * 100);
```

Дело в том, что вещественные числа в компьютере могут представляться точно, с избытком или с недостатком (см., например, книгу Е. Андреевой, И. Фалиной «Системы счисления и компьютерная арифметика», М.: Бином. Лаборатория знаний, 2004). Грубо говоря, вместо 0.35 в компьютере может храниться число 0.349999999… Поэтому умножение на 100 и отбрасывание дробной части может привести к неверному результату: 34 вместо 35. В данной задаче для решения этой проблемы можно использовать функцию `round`, но в общем случае это не так. Для получения целого числа рублей функцию `trunc` использовать можно.

10. Задачу проще всего решать, сразу переведя оба момента времени в секунды. В Borland Pascal при этом надо использовать тип данных `longint` не только для результата, но и для описания входных значений часов, минут и секунд.

Задачи повышенной сложности

1. Если значение только одного из введенных чисел равно 0, то результатом должна быть 1 (0 делится на любое другое число), а если оба числа равны 0, то ответ должен быть иным (0 на 0 не делится). Для того что бы избежать деления на 0, добавим к делителю n следующее выражение: 1 div $(1+|n|)$. Оно равно 1 при $n=0$ и 0 — в любом другом случае.

2. Для произвольных целых чисел с использованием модуля решение имеет вид $1+(m-n)-|m-n|$. Задача становится интереснее, если входными данными будут только натуральные числа, а использование модуля будет запрещено.

3. Приведем основной фрагмент решения данной задачи:

```
a1 := a mod 10;
a2 := a div 10 mod 10;
a3 := a div 100 mod 10;
a4 := a div 1000;
k := 1 div (abs(a1 - a2) + 1) + 1 div (abs(a1 - a3) + 1) +
     1 div (abs(a1 - a4) + 1) + 1 div (abs(a2 - a3) + 1) +
     1 div (abs(a2 - a4) + 1) + 1 div (abs(a3 - a4) + 1);
writeln(k);
```

5. За каждую минуту часовая стрелка проходит 0,5°, а минутная — 6°. В момент их совпадения (n_0 часов m_0 минут) будет выполняться следующее равенство: $30n_0+0{,}5m_0=6m_0$ или $60n_0=11m_0$. Поэтому, если в рамках текущего часа минутная стрелка еще не обогнала часовую ($60n_0 \geqslant 11m_0$), то ответом на вопрос задачи будет выражение $60n \operatorname{div} 11-m$, в противном же случае стрелки встретятся уже во время следующего часа, и ответ выглядит так: $60(n+1) \operatorname{div} 11+60-m$. В окончательном решении производится совмещение этих формул (см. решение задачи 7):

```
S := 60*(n + 1 div(1 + (60*n + 1)div(11*m + 1))*12)div 11 - m;
```

7. С использованием модуля задача решается аналогично задаче 2. Например, минимальное из двух чисел выражается так:

$((m+n)-|m-n|)/2$.

Для натуральных чисел возможно решение, не содержащее обращение к модулю:

```
a := m div n;
b := 1 div (a + 1); {b = 1 при m < n и b = 0 при m >= n}
min := n * (1 - b) + m * b;
max := m * (1 - b) + n * b;
```

8. Для всех чисел, отличных от 0, ответ выглядит как $x/|x|$. Для учета нуля воспользуемся практически тем же выражением, что и для целых чисел: $[1/(1+|x|)]$. Оно равно единице при $x=0$ и нулю для всех остальных значений x. Следует также помнить, что результат должен быть присвоен переменной целого типа. В результате получаем:

```
sign := trunc(x/(abs(x) + trunc(1/(abs(x) + 1))));
```

Урок 4

Задачи

1. Границы заштрихованных фигур в данном задании не анализируются сознательно. Это позволяет использовать логическую операцию `not` при описании части плоскости, находящейся по другую сторону от границы фигуры. Так, точки, лежащие внутри единичного круга с центром в начале координат, описываются неравенством $x^2+y^2 \leqslant 1$. Если логической переменной `c` присвоить соответствующее логическое выражение `c := x * x + y * y <= 1`, то `not c` будет соответствовать точкам, лежащим строго вне круга. Однако, если границу круга учитывать, такое использование переменной `c` для описания внешней области круга, становится неправомерным.

Еще следует обратить внимание на порядок выполнения операций. Заметим, что в различных языках программирования этот порядок различный. Например, в языке C операции сравнения имеют больший приоритет, чем логические связки и скобки в логических выражениях не нужны. В языке Pascal без скобок в подобных выражениях не обойтись. В противном случае можно получить не только выражение с синтаксической ошибкой, но и синтаксически верное выражение, имеющее другой смысл. Так, в выражении с целочисленными переменными a и b

```
not a < b
```

сначала будет вычислено побитовое отрицание числа a, а затем выполнена операция сравнения полученного результата с числом b, хотя скорее всего в данном случае подразумевалось выполнение отрицания результата операции сравнения.

Приведем программу, решающую поставленную задачу:

```
var x, y: real;
  incircle, upline1, upline2, answer: boolean;
begin
  readln(x, y);
  incircle := (x + 0.5)*(x + 0.5) + (y - 1)*(y - 1) <= 2 * 2;
```

```
  upline1 := y >= 5 * x / 3 + 2;
  upline2 := y >= -3 * x / 2;
  answer := incircle and upline2 and not upline1 or
         upline1 and not upline2 and not incircle;
  writeln(answer)
end.
```

3. Приведем фрагменты решения для каждого из требуемых вариантов решения:

1)
```
if a < b then
  if b < c then writeln(a, b, c)
  else {a < b, c < b}
    if a < c then writeln(a, c, b)
             else writeln(c, a, b)
else {b < a}
  if a < c then writeln(b, a, c)
  else {b < a, c < a}
    if b < c then writeln(b, c, a)
             else writeln(c, b, a)
```

2)
```
if a > b then
begin
  d := a; a := b; b := d
end;
if b > c then
begin
  d := c; c := b; b := d
end;
if a > b then
begin
  d := a; a := b; b := d
end;
```

4. Общая схема решения такая: сначала задача решается для общего случая, затем анализируется, когда полученное решение некорректно (при $a = 0$). Затем уравнение переписывается уже с нулевым значением a: $b = 0$. Проанализировать последний случай также следует аккуратно. Наконец, обратите внимание, что в задаче требуется найти только целочисленное решение.

Приведем основной фрагмент решения задачи:

```
if a <> 0 then
  if b mod a = 0 then writeln(b div a)
                 else writeln('no solution')
```

```
else if b = 0 then writeln('many solutions')
              else writeln('no solution');
```

6. В данной задаче наиболее сложная часть — построить компактное логическое выражение, описывающее нахождение двух фигур на одной диагонали. Приведем основной фрагмент решения задачи:

```
if (abs(k - m) = abs(l - n)) or (k = m) or (l = n) then
  writeln('YES')
else
  writeln('NO');
```

7. Решение основано на том, что сумма координат клеток одного цвета обладает одной и той же четностью (для одних клеток эта сумма четна, а для клеток другого цвета — нечетна):

```
if (k + l) mod 2 = (m + n) mod 2 then writeln('YES')
                                 else writeln('NO');
```

9. Заметим, что достаточно найти две минимальных стороны кирпича, упорядочить D и E по возрастанию и проверить, что меньшая сторона не больше D, а вторая по величине сторона не больше E.

10. Найдем сначала максимальную из трех сторон и обозначим ее a. После этого в решении фактически анализируется знак косинуса угла, лежащего напротив стороны a:

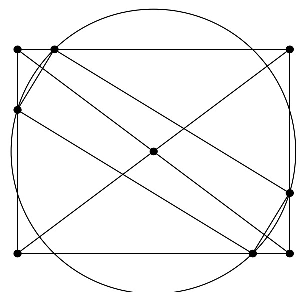
```
if a >= b + c then
  writeln('impossible')
else if a * a = b * b + c * c then
  writeln('rectangular')
else if a * a < b * b + c * c then
  writeln('acute')
else
  writeln('obtuse');
```

11. Рассуждать проще всего так. Если $a \neq 0$, то для нахождения корней используются известные формулы, при этом существование корней зависит от знака дискриминанта квадратного уравнения (проверка точного равенства дискриминанта нулю корректна только для целых коэффициентов уравнения). Если же $a = 0$, то уравнение становится линейным. Однако при $b = 0$ оно вырождается в равенство $c = 0$. Поэтому, если c действительно равно 0, то все действительные числа являются корнями данного уравнения, в противном случае — корней нет.

Задачи повышенной сложности

1. Достаточно упорядочить размеры каждой из коробок так, чтобы $A \leqslant B \leqslant C$, а затем проверить, что $A_1 \leqslant A_2$, $B_1 \leqslant B_2$ и $C_1 \leqslant C_2$ или $A_2 \leqslant A_1$, $B_2 \leqslant B_1$ и $C_2 \leqslant C_1$.

3. Несмотря на сходство с уже рассмотренной задачей 9, решение этой задачи существенно сложнее. Теперь мы можем произвольно поворачивать кирпич относительно сторон отверстия. Можно доказать, что одна из граней кирпича должна быть параллельна плоскости отверстия. Сначала как и в задаче 9 надо проверить, нельзя ли разместить кирпич параллельно сторонам отверстия. После этого надо совместить центр кирпича и центр наименьшей грани (а именно, той грани, стороны которой минимальны) и провести окружность, радиус которой равен половине длины диагонали стороны кирпича (см. рис.).



Точки пересечения данной окружности со сторонами отверстия и определяют прямоугольник (несложно доказать, что полученный четырехугольник — прямоугольник), стороны которого должны быть не меньше соответствующих сторон кирпича. Обратите внимание, что стороны полученного прямоугольника не параллельны ни одной из диагоналей отверстия.

4. Одно из наиболее коротких решений основано на анализе значений скалярных произведений векторов, образующих каждый из углов треугольника (их также можно рассматривать, как характеристику косинуса соответствующего угла). Исключение составляет случай расположения трех различных точек на одной прямой — в данном случае надо проверить коллинеарность двух векторов, причем не используя операцию деления (здесь очень удобно использовать величину векторного произведения, но с подобным понятием школьники скорее всего не знакомы). При определении типа треугольника используются те же идеи, что и в решении задачи 10 (см. выше).

5. Сначала решим задачу для общего случая: умножим первое уравнение на d, а второе на b, вычтем из одного другое и найдем значение x, затем умножим первое из данных уравнений на c, а второе — на a и легко найдем значение y (решение, в котором одна из переменных выражается из первого уравнения и подставляется во второе, существенно сложнее). Из полученных формул видно, что подобное решение невозможно лишь когда $ad - bc = 0$. В последнем случае решений будет или бесконечно много, или не будет совсем. Разобрать получающиеся случаи помогает графическая интерпретация решения данной системы уравнений: результатом является либо точка, либо прямая, либо вся плоскость, либо пустое множество точек.

Урок 5

Задачи

2. Как сказано в условии предыдущей задачи, операция возведения в степень в языке Pascal отсутствует. И это не случайно. Дело в том, что для реализации этой операции в других языках программирования используется тождество: $x^y = e^{y \ln x}$, т. е. фактически запрограммировано сначала вычисление натурального логарифма, а потом — экспоненты. И то и другое представляет собой суммирование соответствующего ряда. Для целых степеней это неэффективно. Так как алгоритм эффективного решения задачи изложен в ее условии, просто приведем его реализацию. Заметим, что все операции умножения следует выполнять по модулю p.

```
readln(x, n, p);
r := 1 mod p; {эта операция необходима для случая p = 1}
q := x;
m := n;
while m > 0 do
begin
  if odd(m) then
    r := r * q mod p
  else
  begin
    m := m div 2;
    q := q * q mod p
  end
end;
writeln(r);
```

3. Это базовая задача для целого класса различных задач, связанных с выделением цифр из числа. Приведем основную часть ее решения без комментариев:

```
readln(n);
m := n;
s := 0;
while m > 0 do
begin
  k := m mod 10;
  m := m div 10;
  s := s + k
end;
writeln(s);
```

4. Для чисел 1, 2, а также всех четных чисел ответ можно выписать сразу. Проверку нечетного числа на простоту будем осуществлять с помощью деления на числа, меньшие n. Обратите внимание, что перебор возможных делителей следует осуществлять только до $\sqrt{n}$ (докажите это самостоятельно).

```
if (n = 1) or (n mod 2 = 0) then writeln('NO')
else if n = 2 then writeln('YES')
else
begin
  prime := true;
  i := 3;
  while i * i <= n do
  begin
    if n mod i = 0 then
      prime := false;
      i := i + 2
  end;
  if prime then writeln('YES')
           else writeln('NO')
end;
```

5. Будем искать разложение n вида $n=i^2+j^2$, где $i \leqslant j$. В этом случае $i \leqslant \sqrt{(n/2)}$. Далее достаточно перебрать все возможные значения i, начиная с 1, и вычислить соответствующее значение j по формуле. Затем проверить, действительно ли выполняется требуемое равенство. При решении большое внимание следует обращать на точность вычислений и правомочное округление вещественных чисел.

7. Заметим, что при отсутствии округления при вычислении процентов можно было бы не указывать начальную сумму вклада — любая сумма удваивалась бы за один и тот же срок и вычислить его можно было бы по формуле. В случае с округлением это не всегда так. Поэтому будем моделировать увеличение средств на счете, пока сумма не удвоится. Очевидно, что для этого деньги на счету должны пролежать как минимум год, поэтому для решения больше подходит цикл `repeat` (для задачи 6 — `while`):

```
readln(n, k);
ans := 0;
m := 2 * n;
repeat
  n := n + n * k div 100;
  ans := ans + 1
until n >= m;
writeln(ans);
```

8. Алгоритм Евклида является одним из важнейших целочисленных алгоритмов. Приведем наиболее короткую его реализацию:

```
read(n, m);
while m > 0 do
begin
  r := n mod m;
  n := m;
  m := r
end;
writeln(n);
```

Обратите внимание, что в решении не нужно проверять, какое из чисел больше — n или m. Если после считывания данных $n < m$, то после первого выполнения оператора цикла значения просто поменяются местами. Приведенная программа верно работает и когда одно из входных значений равно 0.

9. Эта задача также является очень важной при изучении программирования. Она относится к классу задач на обработку числовых последовательностей без использования массивов. Кроме того, при ее решении используются такие базовые алгоритмы, как поиск максимального и минимального значения элементов последовательности. Приведем основной фрагмент решения задачи:

```
read(a);
min := a;
max := a;
```

```
while a <> 0 do
begin
  if a < min then min := a
  else if a > max then max := a;
  read(a)
end;
writeln(min,' ',max);
```

Обратите внимание, что если все значения элементов последовательности считывать в цикле, а начальные значения минимума и максимума проинициализировать заведомо большим и заведомом маленьким числом соответственно, то программа может неправильно работать, например, на убывающей последовательности.

11. Так как массив в программе использовать нельзя, будем хранить в двух переменных соседние элементы входной последовательности. Будем также считать, что помимо последнего (барьерного) элемента, который не надо учитывать в решении, в последовательности есть по крайней мере 2 элемента. Решение может выглядеть так:

```
ac  := true;
wac := true;
dc  := true;
wdc := true;
cn  := true;
read(a, b);
while b <> -2000000000 do
begin
  if a >= b then ac  := false;
  if a >  b then wac := false;
  if a <= b then dc  := false;
  if a <  b then wdc := false;
  if a <> b then cn  := false;
  a := b;
  read(b)
end;
if ac  then writeln('ASCENDING')  else
if dc  then writeln('DESCENDING') else
if cn  then writeln('CONSTANT')   else
if wac then writeln('WEAKLY ASCENDING') else
if wdc then writeln('WEAKLY DESCENDING')
       else writeln('RANDOM');
```

Обратите внимание, что после анализа последовательности проверять признаки на истинность в произвольном порядке нельзя. Так, если последовательность является слабо возрастающей, то она же может оказаться и постоянной, обратное же неверно, поэтому признаки слабого возрастания и слабого убывания проверяются в последнюю очередь.

Урок 6

Задачи

1. В наиболее естественном решении знаменатель очередного слагаемого вычисляется через значение параметра цикла:

```
k := 1;
s := 1;
for i := 1 to n do
begin
  k := -k;
  s := s + k/(2 * i + 1)
end;
```

3. Отсутствие массива в решении указывает в этой задаче на то, что решить ее нужно за один просмотр данных (решение за два просмотра очевидно). Тем не менее, и при таких ограничениях решение не становится намного сложнее:

```
read(n);
read(a);
min := a;
kol := 1;
for i := 2 to n do
begin
  read(a);
  if a < min then
  begin
    min := a;
    kol := 1
  end
  else if a = min then
    kol := kol + 1
end;
write(kol);
```

5. В решении задачи предполагается, что количество судей было не меньше трех. В остальном решение достаточно очевидное: по ме-

ре считывания данных будем искать минимальное и максимальное значения среди всех оценок, а также сумму всех без исключения оценок. Для расчета балла спортсмена достаточно из общей суммы вычесть минимальное и максимальное значения, при этом неважно, какие именно судьи их поставили.

7. В задачах на суммирование числовых рядов важно научиться определять, как изменяется очередное слагаемое (или его части) по сравнению с предыдущим. Приведем решение, знак слагаемого пересчитывается отдельно:

```
read(n, x);
ans := 1;
  s := 1; {очередное слагаемое}
  k := 1; {знак слагаемого}
for i := 1 to n do
begin
  s := s * x * x / ((2 * i - 1) * 2 * i);
  k := -k;
  ans := ans + k * s
end;
writeln(ans:0:6);
```

Подобные ряды можно суммировать и при бесконечном количестве слагаемых (ряд при этом должен сходиться). Процесс вычислений тогда заканчивается, когда очередное слагаемое оказывается меньше интересующей нас точности вычислений (но мы не можем утверждать, что получили сумму ряда с такой же точностью, для этого нужно получить оценку на сумму неучтенных слагаемых).

8. В виде рекуррентного соотношения условие задачи можно записать так: $s_i = \sqrt{mi + s_{i+1}}$; $s_{n+1} = 0$. Требуется найти s_1. Программа для решения этой задачи оказывается совсем простой, и на ее примере можно вспомнить правила использования цикла с downto:

```
read(n, m);
ans := 0;
for i := n downto 1 do
  ans := sqrt(i * m + ans);
writeln(ans:0:6);
```

9. Числа Фибоначчи и в математике, и в программировании используются достаточно часто. Для того чтобы решить задачу без использования массива, нам придется завести три переменных для хранения трех соседних чисел Фибоначчи и аккуратно их переопределять по мере вычислений. Заметим, что 40-е число Фибоначчи заведо-

мо невозможно найти с использованием типа `integer` языка Borland Pascal. Кроме того, необходимо следить за тем, чтобы программа корректно находила и первое и нулевое числа Фибоначчи:

```
read(n);
f1 := 1;
f2 := 1;
for i := 2 to n do
begin
  f := f1 + f2;
  f2 := f1;
  f1 := f
end;
if n > 1 then writeln(f)
         else writeln(1);
```

10. Единственная сложность при решении данной задачи — аккуратная обработка входной последовательности за один просмотр:

```
read(n);
mn1 := 31000;
{можно использовать любые значения, большие 30000}
mn2 := 31000;
for i := 1 to n do
begin
  read(a);
  if a <= mn1 then
  begin
    mn2 := mn1;
    mn1 := a
  end
  else if a < mn2 then
    mn2 := a
end;
writeln(mn1, ' ', mn2);
```

11. Без массива данную задачу можно решать с помощью схемы Горнера только в случае, когда коэффициенты полинома задаются в определенном порядке: от коэффициента при максимальной степени x (a_n) до коэффициента при нулевой степени (a_0):

```
read(n, x0);
ans := 0;
for i := n downto 0 do
begin
```

```
  read(a);
  ans := ans * x0 + a
end;
writeln(ans:0:3);
```

Урок 7

Задачи

1. Программу можно написать так, что и без применения условных операторов будут генерироваться все необходимые тройки:

```
read(n);
for i := 1 to n div 3 do
  for j := i to (n - i) div 2 do
    ans := ans + 1;
writeln(ans);
```

А теперь обратим внимание, что внутренний цикл фиксированное число раз добавляет к ответу 1. Его можно заменить на один оператор присваивания:

```
ans := ans + (n - i) div 2 - i + 1;
```

2. а) Ответом на задачу является число 52251.

б) Основная идея решения. Подсчитаем сумму цифр у левой половины номера. Счастливыми окажутся все билеты, у которых сумма правых цифр будет такой же. Так, для шестизначного билета, если мы обозначим количество трехзначных чисел, сумма цифр у которых равна, например, 15, за S_{15}, то количество счастливых билетов, сумма первых трех цифр которых равна сумме трех последних и равна 15, выражается формулой $S_{15} \times S_{15}$. Общее число счастливых билетов для шестизначных билетов равно $S_1^2 + S_2^2 + \ldots + S_{27}^2$. Для произвольных четных n ответ получается аналогично. Для нечетных n ответ получается домножением ответа, полученного для $n-1$, на 10 (средняя цифра может быть любой, в том числе и нулевой) и прибавить 9 (билеты, у которых все цифры, за исключением средней, равны 9). В этом случае однозначные билеты не являются исключением. Их количество просто равно 9, и все они счастливые. Приведем возможную реализацию приведенного решения:

```
var n, i, s,
  a1, a2, a3, a4, a5, a6,
  n1, n2, n3, n4, n5, n6: integer;
  a, ans: extended;
```

```
begin
  read(n);
  if n > 3  then n1 := 9 else n1 := 0;
  if n > 5  then n2 := 9 else n2 := 0;
  if n > 7  then n3 := 9 else n3 := 0;
  if n > 9  then n4 := 9 else n4 := 0;
  if n > 11 then n5 := 9 else n5 := 0;
  if n > 13 then n6 := 9 else n6 := 0;
  ans := 0;
  for s := 1 to (n div 2) * 9 do
    begin
      a := 0;
      for a1 := 0 to n1 do
      for a2 := 0 to n2 do
      for a3 := 0 to n3 do
      for a4 := 0 to n4 do
      for a5 := 0 to n5 do
      for a6 := 0 to n6 do
        if s - a1 - a2 - a3 - a4 - a5 - a6 in [0..9]
          then a := a + 1;
      ans := ans + a * a
    end;
  if n mod 2 = 1 then ans := ans * 10 + 9;
  writeln(ans:0:0);
end.
```

Заметим, что с использованием массивов (n_1[1..7], в котором бы указывалось количество цифр в соответствующем разряде, и s[1..63], где $s[i]$ — количество $[n/2]$-значных чисел, сумма цифр которых равна i) решение было бы короче. Обратите также внимание, что ответ не умещается в целый тип `longint`, поэтому для нахождения результата используется вещественный тип `extended` (в языке Delphi можно было бы использовать тип `int64`). Для больших значений n задачу нужно решать с помощью динамического программирования.

3. При данных ограничениях задача решается перебором всех возможных вариантов выдачи сдачи. Однако после того, как мы зафиксировали количество 10-рублевых купюр и монет в 5 и 2 рубля, количество рублевых монет определяется однозначно, поэтому мы перебираем только три старших номинала денежных единиц:

```
read(n);
ans := 0;
for i := 0 to n div 10 do
```

```
  for j := 0 to (n - i * 10) div 5 do
    for k := 0 to (n - i * 10 - j * 5) div 2 do
      ans := ans + 1;
writeln(ans);
```

Далее следует применить прием, приведенный в решении задачи 1, и избавиться от внутреннего цикла.

4. Задача относится к числу базовых проблем, которые можно рекомендовать для обязательного рассмотрения. Алгоритм решения задачи следующий. Возьмем первое простое число — 2. Пока n делится на 2, то будем делить его на 2 и двойки сразу печатать. Если число на 2 уже не делится, то будем пытаться делить его на 3 и т. д. Интересно, что проверять при этом делители на простоту не нужно! Дело в том, что если число уже разделили на 2 и на 3 максимально возможное число раз, то ни на 4, ни на 6, ни на 9 оно не разделится автоматически. Поэтому делители можно проверять подряд, но делать это только до квадратного корня из исходного числа n (при программировании следует использовать выражение вида `i * i <= n`). Если оставшееся число больше единицы, то оно тоже простое и его нужно напечатать.

Основную техническую сложность представляет аккуратный вывод результата. Печать очередного найденного делителя может выглядеть так:

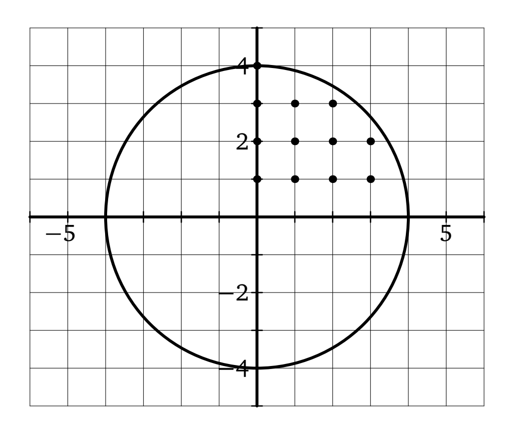
```
write(i);
if n > 1 then write('*');
```

Последнее условие означает, что у рассматриваемого числа n есть еще по крайней мере один простой делитель.

7. Перебрать все точки в данном случае не удастся. Решим задачу для четверти круга. Целочисленные точки будем суммировать по столбцам. На оси координат Oy выше оси Ox лежит ровно r целочисленных точек круга. В каждом следующем столбце будет либо столько же целочисленных точек круга, либо меньше (см. рис.). На сколько меньше — проще всего выяснить, вычитая точки по одной и подставляя результат в уравнение окружности (на самом деле, в неравенство, которое позволяет определить, лежит ли точка внутри круга). В этом случае все вычисления будут производится точно (но при больших r придется использовать тип `int64`). Количество точек в столбце можно определять и по формуле, но тогда могут возникнуть проблемы с точностью.

Приведем возможный вариант соответствующей программы:

```
readln(r);
ans := r;
```




```
s := r;
for i := 1 to r - 1 do
begin
  while r * r < i * i + s * s do
    s := s - 1;
  ans := ans + s
end;
ans := 4 * ans + 1;
{начало координат учитывается отдельно}
writeln(ans);
```

9. Задачу можно рассматривать, как проблему для проведения вычислительного эксперимента. Сначала определим диапазоны значений факториала, для которых вычисления можно проводить точно в каждом из перечисленных типов, а потом напишем диалоговую программу, решающую поставленную задачу.

Для значений целых типов ответ получить не сложно — допустимо сравнивать результат вычислений в таком типе с результатом вычислений в типе `extended`. Для анализа последнего достаточно заметить, что при последовательном получении факториалов после домножения 22! на 23 у результата появляется лишний ноль, чего не может быть. Соответственно максимальное n, для которого в стандартной компьютерной арифметике можно вычислить факториал точно, равно 22.

Часто возникает вопрос, почему в типе `extended` результат можно вычислить для большего значения n, нежели для типа `int64`. Ведь в первом из них на хранение мантиссы (цифр числа) отводятся те же 64 бита. Дело в том, что факториалы, записанные в двоичной системе

счисления, начиная с 2! имеют на конце несколько нулей. Например, $5! = 120 = 1111000_2$. Поэтому в мантиссе будут записываться двоичные цифры, начиная с правой единицы, а количество нулей в конце числа будет учтено в его порядке.

10. Задача отличается от задачи 9 предыдущего урока тем, что количество корней теперь не органичено. Однако, если указанное выражение сходится к конечному числу, то, начиная с некоторого количества корней, требуемое количество цифр результата меняться уже не будут. Поэтому мы можем каждый раз решать конечную задачу заново для различного числа корней, пока не будет достигнута требуемая точность:

```
read(m);
ans1 := 0;
ans2 := 1;
n := 5;
while ans2 - ans1 > 1e-7 do
begin
  ans1 := ans2;
  ans2 := 0;
  for i := n downto 1 do
    ans2 := sqrt(i * m + ans2);
  n := n + 1
end;
writeln(ans2:0:6);
```

Урок 8

Задачи

1. При решении этой задачи основные сложности вызывает выдача на экран именно по 60 символов в строке. Перевод строки следует производить после 59-го, 119-го, 179-го и т. д. символов, естественно, не указывая их явным образом в программе. Перевод строки, осуществляющийся в результате печати соответствующего управляющего символа, при этом учитывать не нужно. Приведем фрагмент одного из возможных решений:

```
for c := #0 to #255 do
begin
  write(c);
  if (ord(c) + 1) mod 60 = 0 then writeln
end;
```

3. Цифры шестнадцатеричных чисел в языке Pascal можно считать только в символьный тип, а потом перевести в числовой аналог:

```
var i, s1, s2: integer;
  c1, c2: char;
begin
  readln(c1, c2);
  c1 := upcase(c1);
  c2 := upcase(c2);
  if c1 in ['0'..'9'] then
    s1 := ord(c1) - ord('0') else
  if c1 in ['A'..'F'] then
    s1 := ord(c1) - ord('A') + 10;
  if c2 in ['0'..'9'] then
    s2 := ord(c2) - ord('0') else
  if c2 in ['A'..'F'] then
    s2 := ord(c2) - ord('A') + 10;
  writeln(16 * s1 + s2)
end.
```

4. В наиболее элегантном решении данной задачи не используются числовые переменные и даже знание того, что в латинском алфавите 26 букв:

```
var c1, c2: char;
begin
  for c1 := 'a' to 'z' do
  begin
    for c2 := 'a' to c1 do
      write(c2);
    writeln
  end
end.
```

6. Если указанный в условии формат входных данных строго соблюдается и, кроме запятых, между словами нет других символов, например, пробелов, то задача решается достаточно легко:

```
k := 0;
repeat
  read(c);
  if upcase(c) = 'A' then k := k + 1;
  {пропуск остальных букв слова}
  repeat
    read(c)
```

```
  until (c = ',') or (c = '.')
until c = '.';
writeln(k);
```

7. Решение этой задачи дает единый ключ к пониманию того, как грамотно распечатать результат работы программы на русском языке. Это касается, например, и денежных расчетов (*рубль*, *рубля*, *рублей*), причем правила выбора нужного падежа остаются неизменными. Несложно заметить, что в основном определяющей здесь является последняя цифра числа. Однако исключение составляют числа, дающие при делении на 100 остатки 11..14, причем в нашей задаче это касается и чисел 111..114 (про них очень часто забывают). Приведем основной фрагмент решения задачи:

```
if k mod 100 in [10..20] then writeln('Мне ', k, ' лет')
else
  case k mod 10 of
    0, 5..9: writeln('Мне ', k, ' лет');
          1: writeln('Мне ', k, ' год');
       2..4: writeln('Мне ', k, ' года');
  end;
```

9. Задача отрабатывает умение работать с перечислимыми типами данных, в том числе решать вопросы ввода и вывода соответствующих значений, а также требует уверенного использования операторов case. Полезно решить ее так, чтобы разбор вариантов происходил непосредственно для констант перечислимого типа, а не для их числовых аналогов. В данном случае необходимо использовать вложенные операторы варианта, причем для получения компактной программы важно, какой из операторов будет внешним, а какой — внутренним. Приведем недостающий фрагмент решения задачи:

```
  case p of
    forwrd: k2 := k1;
    left: if k1 = north then k2 := west
                        else k2 := pred(k1);
    right: if k1 = west then k2 := north
                        else k2 := succ(k1);
    back: case k1 of
            north,east: k2 := succ(succ(k1));
            south,west: k2 := pred(pred(k1));
          end
  end;
```

10. Задача отрабатывает умение писать вложенные циклы, параметром которых являются логические переменные. Подобная программа может также пригодиться при решении логических задач. Приведем программу печати таблицы истинности полностью:

```
var a, b, c: boolean;
begin
  writeln('   a      b      c   f(a, b, c)');
  for a := false to true do
    for b := false to true do
      for c := false to true do
        writeln(a:7, b:7, c:7, a or not b and c:7)
end.
```

Печать по формату в данном случае нужна для выравнивая логических констант в столбцах.

Урок 9

Задачи

1. Первые две задачи помогают освоить заполнение массива случайными числами из требуемого диапазона. Приведем их решения полностью.

а)
```
var a, b, i: integer;
c: array[1..100] of integer;
begin
  readln(a, b);
  for i := 1 to 100 do
    c[i] := a + random(b - a + 1);
  for i := 1 to 100 do write(c[i],' ')
end.
```

б)
```
var i: integer;
c: array[1..100] of real;
begin
  for i := 1 to 100 do c[i] := 10 * random;
  for i := 1 to 100 do write(c[i]:0:4,' ')
end.
```

2. Несмотря на то что аналогичная задача приведена в материалах урока, решение этой задачи часто вызывает затруднение. Обычно предлагается следующий неверный вариант:

```
c := a[n];
for i := 2 to n do
```

```
  a[i] := a[i - 1];
a[1] := c;
```

Реализуйте эту программу на компьютере, а потом объясните результаты ее работы. Правильное решение просматривает элементы в обратном порядке:

```
c := a[n];
for i := n downto 2 do
  a[i] := a[i - 1];
a[1] := c;
```

3. Алгоритм поиска максимального и минимального элементов массива по реализации несколько отличается от аналогичного алгоритма для последовательностей. Оказывается, в массиве правильнее искать не значение максимального или минимального элемента, а индексы соответствующих элементов, тогда автоматически мы будем знать и сами значения. Таким образом делать двойную работу не нужно (в одной переменной хранить значение максимального элемента, а в другой — ее индекс). Приведем основную часть наиболее экономичного решения задачи.

```
imax := 1; {индекс максимального элемента}
imin := 1; {индекс минимального элемента}
for i := 2 to n do
  if a[i] > a[imax] then imax := i else
  if a[i] < a[imin] then imin := i;
c := a[imin];
a[imin] := a[imax];
a[imax] := c;
```

6. Эта задача фактически отрабатывает идею «подсчета» — встретив очередную букву, будем учитывать ее в соответствующем элементе массива. Задачи на подобную идею часто встречаются на ЕГЭ по информатике (часть C). Кроме того, на примере этой задачи можно наглядно показать, что индексировать элементы массивов иногда очень удобно нечисловыми порядковыми величинами. Обратите внимание, что при правильном решении данной задачи текст просматривается ровно один раз, а массив результатов заполняется не по порядку, а по мере прочтения текста. Программа должна также учитывать, что в тексте могут встречаться символы, отличные от букв. Приведем основной фрагмент решения задачи:

```
for c := 'A' to 'Z' do b[c] := 0;
for i := 1 to 1000 do
  if a[i] in ['A'..'Z', 'a'..'z'] then
```

```
  begin
    c := upcase(a[i]);
    b[c] := b[c] + 1
  end;
```

8. Искомыми числами являются либо два максимальных элемента массива, либо два минимальных (это возможно при наличии как минимум двух отрицательных чисел). Поэтому сначала найдем эти 4 числа, а затем выясним, произведение какой пары чисел больше. Сделать это можно за один проход исходного массива (см. решение задачи 10 урока 6).

9. При заданных ограничениях никакая эффективность при решении задачи не требуется, проще всего каждый элемент проверить на уникальность. Типичная ошибка: чтобы проверить, не равен ли рассматриваемый элемент какому-то другому, он сравнивается со всеми элементами массива, а следовательно, и с самим собой, что может привести к неверному результату. Приведем вариант правильного решения:

```
for i := 1 to n do
begin
  cnt := 0;
  for j := 1 to n do
    if a[i] = a[j] then cnt := cnt + 1;
  if cnt = 1 then write(a[i], ' ')
end;
```

10. Задача лишь немного сложнее предыдущей. Приведем очевидное, хотя и не самое короткое решение, использующее булевские переменные:

```
for i := 1 to n do
begin
  f1 := true;
  for j := 1 to i - 1 do
    if c[j] = c[i] then
      f1 := false; {элемент уже был распечатан}
  f2 := false;
  for j := i + 1 to n do
    if c[i] = c[j] then
      f2 := true; {элемент не уникален}
  if f1 and f2 then
    write(c[i],' ')
end;
```

Задачи повышенной сложности

1. При данных ограничениях решение основывается на непосредственной проверке чисел на простоту, однако в качестве делителей рассматриваются только найденные ранее простые числа, не превосходящие корня из проверяемого числа. Причем операцию извлечения корня надо заменить на возведение в квадрат простого числа и сравнения результата с проверяемым числом:

```
var n, i, j, k, l: longint;
  p: array[0..100000] of longint;
begin
  for i := 0 to 100000 do
    p[i] := 0;
  read(n);
  k := 0;
  p[0] := 1;
  i := 2;
  repeat
    j := 0;
    l := 0;
    while (l = 0) and (j < k) and (p[j] * p[j] < i) do
    begin
      j := j + 1;
      if i mod p[j] = 0 then l := 1
    end;
    if l = 0 then {нашли очередное простое число}
    begin
      k := k + 1;
      p[k] := i
    end;
    i := i + 1
  until k = n;
  writeln(p[k])
end.
```

2. Поскольку $n!$ может быть очень велико (см. задачу 8 урока 7), непосредственное вычисление ответа невозможно. Разложим число k на простые множители. Тогда если $k = P_1^{a_1} P_2^{a_2} \cdots P_S^{a_S}$, то если $n!$ делится на соответственно $P_i^{b_i}$, то $n!$ делится на k^Z, где $Z = \min\limits_{i=1..S}\left\{\left\lfloor \frac{b_i}{a_i} \right\rfloor\right\}$. Единственная оставшаяся проблема — как для простого числа P найти максимальную его степень, на которую делится $n!$.

Для решения этой задачи применим следующие соображения: количество чисел, кратных P и не превышающих n, равно $\left\lfloor \frac{n}{P} \right\rfloor$. Каждое из этих чисел даст по одному простому множителю P в $n!$. Но кроме того, $\left\lfloor \frac{n}{P^2} \right\rfloor$ чисел дадут еще по одному P, $\left\lfloor \frac{n}{P^3} \right\rfloor$ — еще по одному и т. д. Значит, $b_i = \left\lfloor \frac{n}{P_i} \right\rfloor + \left\lfloor \frac{n}{P_i^2} \right\rfloor + \left\lfloor \frac{n}{P_i^3} \right\rfloor + \ldots$ Заметим, что суммирование можно остановить, когда очередное слагаемое равно 0.

3. Эта знакомит с такой идеей как «два указателя». Приведем возможный вариант ее решения:

```
j := 0; {количество встретившихся ненулевых элементов}
for i := 1 to n do
  if a[i] <> 0 then
  begin
    j := j + 1;
    if i <> j then
    begin
      a[j] := a[i];
      a[i] := 0
    end
  end;
```

Здесь в переменной j хранится индекс последнего обработанного ненулевой элемента (левый указатель), i — правый указатель.

4. В этой задаче достаточно заметить, как меняется сумма соседних элементов массива при переходе к следующим $k+1$ элементам:

```
for i := 1 to n do
  read(a[i]);
sum := 0;
for i := 1 to k + 1 do
  sum := sum + a[i];
if sum = m then write(1)
else
begin
  for i := 2 to n - k do
  begin
    sum := sum - a[i - 1] + a[i + k];
    if sum = m then
    begin
      write(i); halt
    end
```

```
  end;
  write(0);
end;
```

5. Эта задача подробно разбирается в брошюре Я. Зайдельмана «Эффективность алгоритмов. Простые задачи и наглядные примеры» (М.: Чистые пруды, 2006).

6. Возможно, например, такое решение данной задачи. Будем последовательно просматривать элементы первого массива и считать, сколько элементов, равных рассматриваемому, расположены в первом массиве до него. Затем подсчитаем, сколько элементов с таким же значением во втором массиве. Если их не меньше, то рассматриваемый элемент можно распечатывать. Приведем следующую реализацию этого алгоритма:

```
for i := 1 to n do
begin
  kol := 1;
  for j := 1 to i - 1 do
    if a[j] = a[i] then kol := kol + 1;
  for j := 1 to m do
    if b[j] = a[i] then kol := kol - 1;
  if kol <= 0 then write(a[i], ' ')
end;
```

Другой способ решения основан на том, что элементы массива по модулю не превосходят 10 000. Поэтому мы можем помечать использованные элементы, прибавляя к ним число 20 001. После решения задачи это число надо снова вычесть из значений тех переменных, которые превосходят 10 000.

7. Красивое решение этой задачи предложено в книге А. Шеня «Программирование: теоремы и задачи» (М.: МЦНМО, 2014). Там эта задача формулируется в следующем виде: требуется поменять местами первые $n-k$ элементов с последними k элементами. Решается она так: если мы перевернем весь массив, затем перевернем первые k элементов и, наконец, последние $n-k$ элементов, то получим искомый сдвинутый массив.

8. Эта задача решается за один или два прохода массива. Очевидно, что если рядом стоят несколько одинаковых элементов, то оставить нужно только один из них. Аналогично, если мы имеем строго возрастающую (убывающую) подпоследовательность из подряд идущих элементов нашей последовательности, то все элементы, кроме

минимального и максимального, нужно удалять (на этом участке мы лишний «зубец пилы» сделать не сможем). Например, последовательность 1 2 3 4 3 1 5 преобразуется в 1 4 1 5. Тогда решение может выглядеть так: сначала удалим одинаковые элементы, первый и последний элементы оставим, а среди элементов со второго по $(n-1)$-й оставим только те элементы, которые либо строго больше своих соседей, либо строго меньше них.

9. Для решения этой задачи за один проход массива необходимо использовать такую структуру данных как стек, который в свою очередь организуется на обычном массиве размером совпадающим с исходным. Пока числа в исходном массиве возрастают, они записываются в стек. Как только очередное число окажется меньшее чем число в вершине стека, ответ для числа из вершины стека найден, и оно из стека удаляется. Аналогично поступаем с числами, оставшимися в стеке, пока в вершине не окажется число, меньшее текущего, или стек не станет пустым. Тогда текущее число тоже заносится в стек.

10. Несмотря на простую постановку, решение оказывается не слишком очевидным. Будем перебирать правое из искомых чисел. Тогда наилучшее левое для него — минимальное из стоящих левее. Минимум из первых i элементов легко пересчитать, зная минимум из первых $i-1$ элементов. Если разница между текущим элементом и минимальным левее его оказывается больше, чем была ранее, то мы запоминаем такую пару как текущий ответ.

Урок 10

Задачи

1. а) В этой задаче не требуется при просмотре всего двумерного массива заполнять нужные места символом «*». Это можно сделать с помощью одного цикла (считается, что предварительно весь массив заполнен пробелами):

```
for i := 1 to n do
begin
  a[i, i] := '*';
  a[i, n - i + 1]   := '*';
  a[i, n div 2 + 1] := '*';
  a[n div 2 + 1, i] := '*'
end;
```

2. Как и в случае главной диагонали, сначала нужно выписать уравнение «побочной» диагонали: $j = n - i + 1$, а потом распечатать требуемые элементы:

```
for i := 1 to n do
begin
  for j := 1 to n - i + 1 do write(a[i, j]:2, ' ');
  writeln
end;
```

4. Задача предназначена для отработки умения индексировать соседние элементы матрицы. Но она важна и сама по себе. Полученные числа i-й строки соответствуют числу сочетаний из i элементов по j элементов (где j — номер соответствующего столбца). Несмотря на наличие формулы для нахождения этих чисел, в программировании их достаточно часто находят именно так, как описано в данной задаче. Объясняется это тем, что растут такие числа быстро и для их вычисления приходится применять так называемую «длинную арифметику», и в приведенном алгоритме придется программировать только одно действие — сложение, а для вычисления по формуле — умножение и деление. Приведем решение задачи с использованием двумерного массива (хотя можно ограничиться и двумя одномерными):

```
for i := 2 to n do
  for j := 1 to i do
    a[i, j] := 0;
a[1, 1] := 1;
for i := 2 to n do
  for j := 1 to i do
    a[i, j] := a[i - 1, j] + a[i - 1, j - 1];
for i := 1 to n do
begin
  for j := 1 to i do
    write(a[i, j], ' ');
  writeln
end;
```

6. В задаче возможны «математическое» и «техническое» решения. В первом случае для элементов каждой строки будем вычислять, какие числа следует ставить на каждое из мест. Во втором случае можно обходить двумерный массив «змейкой» и последовательно расставлять числа от 1 до nm. Приведем основную часть первого решения:

```
readln(n, m);
for i := 1 to n do
```

```
begin
  if i mod 2 = 1 then
    for j := 1 to m do
      a[i, j] := (i - 1) * m + j
  else
    for j := 1 to m do
      a[i, j] := (i - 1) * m + m - j + 1
end;
```

7. А для этой задачи приведем техническое решение, использующее «барьерные» элементы, расположенные вокруг основного массива. Идея решения в следующем: будем заполнять матрицу, двигаясь в одном направлении, пока стоящие на пути элементы равны нулю. После этого направление меняется:

```
readln(m,n);
for i := 0 to m + 1 do
  for j := 0 to n + 1 do
    a[i, j] := -1;
{сначала весь массив заполним -1, а потом обнулим
 "настоящие" элементы, таким образом получим барьер из -1}
for i := 1 to m do
  for j := 1 to n do
    a[i, j] := 0;
k := 0;
i := 1;
j := 0;
repeat
  while a[i, j + 1] = 0 do
  begin
    j := j + 1;
    k := k + 1;
    a[i, j] := k
  end;
  while a[i + 1, j] = 0 do
  begin
    i := i + 1;
    k := k + 1;
    a[i, j] := k
  end;
  while a[i, j - 1] = 0 do
  begin
```

```
    j := j - 1;
    k := k + 1;
    a[i, j] := k
  end;
  while a[i - 1, j] = 0 do
  begin
    i := i - 1;
    k := k + 1;
    a[i, j] := k
  end
until k = n * m;
```

8. При возможности использовать дополнительный массив задача становится несложной:

```
for i := 1 to n do
  for j := 1 to n do
    b[j, n - i + 1] := a[i, j];
```

9. Методом «от противного» легко показать, что в матрице, в которой нет одинаковых элементов, если искомый элемент существует, то он единственный. Для ускорения решения задачи полезно ввести дополнительный одномерный массив, в котором для каждого столбца запоминать номер строки, в которой достигается максимум в этом столбце.

Урок 11

Задачи

1. Если не ставить цель написать самую эффективную программу, решающую поставленную задачу, то решение может выглядеть достаточно элегантно:

```
readln(s);
{пока в строке встречаются два пробела подряд,
 удаляем один из них}
while pos('  ', s) > 0 do delete(s, pos('  ', s), 1);
if s[1] = ' ' then delete(s, 1, 1);
if s[length(s)] = ' ' then delete(s, length(s), 1);
```

3. Приведем возможный вариант решения, основанный на том утверждении, что строка-палиндром совпадает со своим же прочтением справа налево. Поэтому для проверки конца входной строки на палиндром удобно иметь исходную строку и в перевернутом виде:

```
var s, s1, s2: string;
    i, j, k: integer;
```

```
begin
  readln(s);
  j := length(s);
  s1 := '';
  for k := j downto 1 do s1 := s1 + s[k];
  {в s1 перевернутая строка s}
  for i := 1 to j do
    if s[i] = s[j] then
    begin
         s2 := copy(s, i, (j - i + 1) div 2);
      if s2  = copy(s1, 1, length(s2)) then
      begin writeln(i - 1); halt end
    end
end.
```

5. Размер входных данных в задаче не ограничен (ограничение приведено только для одного слова), поэтому считывание всех символов сразу в одну строку может оказаться невозможным. Приведем решение задачи с использованием посимвольного считывания:

```
readln(m);
read(c);
n := 0;
s := c;
repeat
  while not eoln and (c <> ' ') do {выделение одного слова}
  begin
    read(c);
    s := s + c
  end;
  if c = ' ' then delete(s,length(s),1);
  if n + length(s) <= m then
{очередное слово умещается в строке}
  begin
    write(s);
    n := n + length(s)
  end
  else
  begin
    writeln; {перейдем к новой строке}
    if n > 0 then delete(s, 1, 1);
    write(s);
```

```
    n := length(s);
  end;
  if not eoln then
  begin
    read(c);
    s := ' ' + c
  end
until eoln;
```

6. Подобная подзадача встречается в задачах ЕГЭ по информатике. Задача решается проще, если в конце исходной строки приписать пробел (фактически добавить «барьерный» элемент). Тогда признаком окончания очередного слова всегда будет являться пробел.

```
readln(s);
s := s + ' ';
max := 0;
cur := 0;
f := false;
for i := 1 to length(s) do
begin
  if s[i] <> ' ' then cur := cur + 1
  else
  begin
    if cur > max then max := cur;
    cur := 0
  end
end;
writeln(max);
```

7. Чтобы количество пробелов (`blanks`) в `n` группах отличалось не более чем на единицу, их число должно быть равно (`blanks div n`) в (`n - blanks mod n`) группах и `blanks div n + 1` в (`blanks mod n`) группах, причем хотя бы в одной группе будет ровно (`blanks div n`) пробелов. Поэтому именно столько пробелов разместим в первой группе, а далее сведем задачу к задаче меньшей размерности: уменьшим на (`blanks div n`) количество пробелов для дальнейшего распределения и на единицу количество групп пробелов. Тогда на последнем шаге `n` будет равно 1 и все оставшиеся пробелы будут напечатаны. Легко показать, что при таком подходе мы получим необходимое разбиение.

8. Заметим, что удалять найденные вхождения из строки (или из ее копии) нельзя, так как тогда не все вхождения образца, например

aaa, могут быть найдены. Кроме того, при удалении подстроки нумерация символов будет нарушена. Если не заботиться об эффективности решения, то можно просто первый символ очередного вхождения подстроки в строку заменять на отсутствующий в ней, например нулевой, а затем искать подстроку заново.

Урок 12

Задачи

1. В задаче можно применить алгоритм бинарного поиска слова в словаре (массиве строк). Приведем здесь реализацию бинарного поиска, немного отличную от предлагавшейся в материалах урока. Она соответствует следующему инварианту: если искомый элемент существует, то он находится на полуинтервале $[L, R)$.

```
var s: array[1..100000] of string[20];
    w: string;
    n, m, L, R, i: integer;
begin
  readln(w);
  readln(n);
  for i := 1 to n do
    readln(s[i]);
  L := 1; R := n + 1;
  while R - L > 1 do
  begin
    m := (L + R) div 2;
    if a[m] > w then R := m
                else L := m
  end;
  if a[L] = w then write('yes') else write('no')
end.
```

2. Поиск значения k нужно начать с элемента a_{n1}. То есть положим сначала номер строки $i = n$, а номер столбца $j = 1$. Если $a_{ij} < k$, то в текущем столбце элемента заведомо уже нет, он может находиться только правее, поэтому j надо увеличить на единицу. Если же $a_{ij} > k$, то элемент не может находиться в текущей строке, а только выше нее — уменьшаем i на 1. Действуем таким образом, пока искомое значение не будет найдено или мы не выйдем за границу массива. Чтобы не проверять каждый раз выход за границу массива, добавим в исходный массив нулевую строку и $(m+1)$-й столбец, заполнив их значением k.

Таким образом, искомое значение всегда будет найдено. В зависимости от того, в какой части массива это произойдет (в основной или барьерной), мы и определим, на самом ли деле такое значение в массиве есть.

3. С помощью логической переменной `f` будем определять, произошел ли хотя бы один обмен после текущего просмотра массива:

```
f := true;
j := 1;
while f do
begin
  f := false;
  for i := 1 to n - j do
    if a[i] > a[i + 1] then
    begin
      f := true;
      x := a[i];
      a[i] := a[i + 1];
      a[i + 1] := x
    end;
  j := j + 1
end;
```

5. Приведем основной фрагмент указанной сортировки:

```
for j := 2 to n do
begin
  i := j;
  x := a[i]; {запоминаем вставляемый элемент}
  while (i > 1) and (x < a[i - 1]) do
  begin
    a[i] := a[i - 1];
    i := i - 1
  end;
  a[i] := x
end;
```

Урок 13

Задачи

1. Решение основывается на том факте, что

$$\text{НОД}(a,\ b,\ c) = \text{НОД}(\text{НОД}(a,\ b),\ c).$$

Как запрограммировать сам алгоритм Евклида, уже было рассказано в решении задачи 8 урока 5. Приведем программу полностью. Заметим, что для решения этой задачи массив заводить не нужно.

```
var a, b, i, n: integer;
function GCD(n, m: integer): integer;
var r: integer;
begin
  while m > 0 do
  begin
    r := n mod m;
    n := m;
    m := r
  end;
  GCD := n
end;
begin
  readln(n);
  read(a);
  for i := 2 to n do
  begin
    read(b);
    a := nod(a, b)
  end;
  writeln(a)
end.
```

3. Задачу можно решить, вписав треугольник в минимально возможный прямоугольник, параллельный осям координат. Тогда площадь треугольника будет равна площади прямоугольника минус площади «лишних» прямоугольных треугольников (в случае тупоугольного треугольника ситуация незначительно меняется). Если полученную формулу упростить, то для любого треугольника окажется верным, например, следующее равенство:

$$S = |(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)|.$$

5. Приведем решение задачи полностью:

```
procedure aaa(b, n: integer);
var i, j, k: integer;
begin
  k := 1;
  for i := 1 to n div 2 + 1 do
```

```
  begin
    for j := 1 to b do
      write(' ');
    for j := 1 to (n - k) div 2 do
      write(' ');
    for j := 1 to k do
      write('*');
    writeln;
    k := k + 2
  end
end;
begin
  aaa(2, 5); aaa(1, 7); aaa(0, 9)
end.
```

7. Для того чтобы в процедуре поменять местами значения переменных, они должны передаваться по ссылке с добавлением слова `var`:

```
procedure swap(var a, b: integer);
begin
  a := a + b;
  b := a - b;
  a := a - b
end;
```

Когда в качестве фактических параметров такой процедуры будет передаваться указатель на одну и ту же переменную, в результате выполнения второго оператора процедуры переменная обнулится и ее значение будет утеряно. Заметим, что и другие способы обмена значений двух переменных местами без использования третьей переменной приводят к такому же результату.

8. Покажем, как решить эту задачу методом трапеций. Разбиваем отрезок $[a;b]$ на n частей, $h=(b-a)/n$ (шаг интегрирования). По формуле вычисления площади трапеций получаем

$$s_0 = h((f(a)+f(b))/2+f(x_1)+f(x_2)+\ldots+f(x_{n-1})),$$

где $x_i = a+ih$, $i=1,\ldots,n-1$. Уменьшаем шаг в два раза: $h_1 = h/2$, $n_1 = 2n$, тогда $s_1 = s_0/2 + h_1(f(x_1)+f(x_3)+\ldots+f(x_{2n-1}))$, то есть вычисления необходимо производить только для нечетных точек $x_i = a+(2i-1)h_1$, $i=1,\ldots,n$. Если $|s_1 - s_0| < \varepsilon$, то s_1 можно считать решением, в противном случае шаг уменьшается еще в два раза, s_2 вычисляется через s_1, сравниваются s_1 и s_2 и т. д. В результате функция может выглядеть так:

```
type func = function(x: real): real;
function integ(f: func; a, b, eps: real): real;
var i, n: integer;
   s0, s, h: real;
begin
  n := 1;
  h := b - a;
  s := (f(a) + f(b)) / 2 * h;
  repeat
    s0 := s;
    h := h / 2;
    sum := 0;
    for i := 1 to n do s := s + f(a + (2 * i - 1) * h);
    s := s0 / 2 + sum * h;
    n := n * 2
  until abs(s - s0) < eps;
  integ := s
end;
```

9. Приведем пример решения задачи методом хорд. Найдем такой отрезок $[a; b]$, что $f(a) \cdot f(b) < 0$ и корень на этом отрезке единственный. Проводим прямую через точки $(a; f(a))$ и $(b; f(b))$. Находим точку пересечения прямой с осью Ox (назовем эту точку c). Если $f(a) \cdot f(c) > 0$, то `a := c`, в противном случае `b := c` (аналогично методу деления пополам). Далее проводим новую прямую, соединяющую точки a и b. Условие окончания алгоритма (проверяется до очередного переприсваивания значения a или b): $|a - c| < \varepsilon$ или $|b - c| < \varepsilon$. Соответствующая функция может выглядеть так:

```
function chord(f: func; a, b, eps: real): real;
var c, fa: real;
begin
  repeat
    fa := f(a);
    c := a + fa * (b - a)/(fa - f(b));
    if fa * f(c) > 0 then
    begin
       fa := c - a; a := c
    end
    else
    begin
       fa := b - c; b := c
```

```
    end
  until abs(fa) < eps;
  chord := c
end;
```

Урок 14

Задачи

1. Эта задача полезна для понимания рекурсии начинающими. Приведем ее решение для натуральных чисел, но и для произвольных целых чисел она имеет решение:

```
function sum(a, b: integer): integer;
begin
  if a = 0 then sum := b
  else sum := sum(a - 1, b + 1)
end;
```

2. Рекурсивную функцию для нахождения чисел Фибоначчи написать чрезвычайно просто, буквально следуя определению этих чисел:

```
function f(n: integer): longint;
begin
  if n in [0, 1] then f := 1
  else f := f(n - 1) + f(n - 2)
end;
```

Нерекурсивная реализация была приведена в разборе задачи 9 урока 6. Рекурсивная версия будет работать намного дольше. Объясняется это тем, что одни и те же числа Фибоначчи в ней будут неоднократно перевычисляться. Так, чтобы найти $f(6)$, приходится находить $f(4)$ и $f(5)$, в свою очередь, $f(5)$ снова будет вычислять $f(4)$. Количество рекурсивных вызовов при этом растет экспоненциально. Нерекурсивный же вариант использует линейный алгоритм.

4. Здесь фактически описан один из вариантов так называемой «быстрой сортировки». Наиболее эффективную ее реализацию можно найти в примерах к Borland Pascal: `bp\examples\dos\qsort.pas`. Но «классическая» реализация не гарантирует, что первый элемент (точнее, элемент, относительно которого остальные элементы делятся на две группы) будет находиться между двумя группами. Требуемую в условии задачи реализацию проще всего организовать с помощью вспомогательного глобального массива (описание вспомогательного массива внутри рекурсивной процедуры может привести к переполнению стека).

5. Возможная реализация функции, проверяющей часть строки на симметричность:

```
function check(var s: string; i, j: integer): boolean;
begin
  if i > j then check := true
  else
    if s[i] <> s[j] then check := false
                    else check := check(s, i + 1, j - 1)
end;
```

6. Приведем пример процедуры, которая будет сразу печатать результат перевода:

```
procedure c(n, p: integer);
var a: integer;
begin
  a := n div p;
  if a > 0 then c(a, p);
  write(n mod p)
end;
```

7. Задача относится к базовым олимпиадным алгоритмам. Она может быть решена, например, так:

```
var i, k, n: integer;
    a, p: array[0..60] of integer;
procedure print(k: integer);
var j: integer;
begin
  for j := 1 to k do
    write(a[p[j]]:4);
  writeln
end; {print}
procedure cnk(n, k: integer);
 procedure gen(i: integer); {заполняет i-е место сочетания}
 var j: integer;
 begin
   if i > k then print(k) {все элементы выбраны}
   else
     for j := p[i - 1] + 1 to n - (k - i) do
     begin
       p[i] := j;
       gen(i + 1)
     end
```

```
 end; {gen}
begin {cnk}
  gen(1)
end;  {cnk}
begin {main}
  readln(n, k);
  for i := 1 to n do a[i] := i;
  {заполнить массив можно и по-другому}
  cnk(n, k)
end.
```

8. Задача решается методом рекурсивной «заливки» каждого из объектов. В приведенном ниже фрагменте программы отсутствуют описания глобальных переменных.

```
procedure mark(i, j, k: integer);
begin
  if a[i, j] = 1 then
  begin
    a[i, j] := k;
    mark(i + 1, j, k);
    mark(i - 1, j, k);
    mark(i, j + 1, k);
    mark(i, j - 1, k)
  end
end;
begin
  read(n, m);
  for i := 0 to n + 1 do
    for j := 0 to m + 1 do a[i, j] := 0;
  for i := 1 to n do
    for j := 1 to m do read(a[i, j]);
  k := 1;
  for i := 1 to n do
    for j := 1 to n do
      if a[i, j] = 1 then
      begin
        k := k + 1;
        mark(i, j, k)
      end;
  writeln(k - 1)
end.
```

В качестве результата выдается число $k-1$, так как первый объект «красится» числом 2, второй — 3 и т. д. Кроме того, вокруг массива формируется барьер из нулей, которые позволяют не проверять выход рекурсии за границу массива.

9. Приведем основную часть программы, в том числе процедуру, которая заполнит двумерный массив соответствующим образом:

```
const dx: array[1..8] of integer = (1, 2, 2, 1,-1,-2,-2,-1);
      dy: array[1..8] of integer = (2, 1,-1,-2,-2,-1, 1, 2);
var a: array[-1..10, -1..10] of integer;
  x, y, n: integer;
procedure rec(i, x, y: integer);
var j: integer;
begin
  if a[x, y] = 0 then
  begin
    a[x, y] := i;
    if i = n * n then print
    else
      for j := 1 to 8 do
        rec(i + 1, x + dx[j], y + dy[j])
    a[x, y] := 0
  end
end;
begin
  read(n);
  for x := -1 to 10 do
    for y := -1 to 10 do
      {заполним массив барьерными элементами}
      a[x, y] := -1;
  for x := 1 to n do
    for y := 1 to n do {обнулим элементы самой доски}
      a[x, y] := 0;
  rec(1, 1, 1)
end.
```

Урок 15

Задачи

2. Подобные задачи встречаются на ЕГЭ по информатике. Приведем решение полностью:

```
var p: record
         name: string;
          sum: integer;
       end;
  c: char;
  n, m: integer;
  b: boolean;
  f, g: text;
begin
 assign(f, 'f.txt'); reset(f);
 assign(g, 'g.txt'); rewrite(g);
 repeat
   p.name:='';
   repeat
     read(f, c);
     p.name := p.name + c
   until c = ' '; {считана фамилия}
   repeat
     read(f, c);
   until c = ' '; {считан класс}
   p.sum := 0;
   n := 0;
   b := true;
   while not seekeoln(f) do
   begin
     read(f, m);
     if m in [2, 3] then b := false
     else p.sum := p.sum + 1;
     n := n + 1
   end;
   readln(f);
   if b and (p.sum >= 4.5 * n) then writeln(g, p.name)
 until seekeof(f);
 close(g)
end.
```

Обратите внимание, на то, что массив для хранения искомых фамилий, а также оценок одного человека не используется. При анализе среднего арифметического деление заменено умножением.

6. Приведем решение чуть более простой задачи, в которой только подсчитываются среднемесячные температуры для каждого месяца.

Обратите внимание на массив констант, который полезно использовать в большинстве задач, связанных с календарем.

```
const d: array[1..12] of integer =
  (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
var m: array[1..12] of real;
  t: real;
  i, j: integer;
  c1, c2: char;
  f, g: text;
begin
  assign(f, 'meteo.dat');
  reset(f);
  assign(g, 'avr.dat');
  rewrite(g);
  for j := 1 to 12 do
    m[j] := 0;
  for i := 1 to 366 do
  begin
    readln(f, c1, c1, c1, c1, c2, t);
    j := (ord(c1) - ord('0')) * 10 + ord(c2) - ord('0');
    m[j] := m[j] + t
  end;
  for j := 1 to 12 do
    writeln(m[j] / d[j]:0:1);
  close(g)
end.
```

7. Подобные задачи проще всего решаются с помощью математического аппарата, называемого «конечным автоматом». В процессе считывания текста автомат находится в различных состояниях, таких как «внутри комментария», «внутри строки» и «программа». В каждом состоянии одни и те же входные символы обрабатываются по-разному. Например, если мы находимся внутри комментария, то автомат просто ждет, когда появится символ, завершающий комментарий. Приведем такое решение задачи:

```
const sr: set of char = [' ', ';', ':', '}'];
var cgoto: word;
  s, s1: string;
  i, j: integer;
  state: (com, str, prog);
  f, g: text;
```

```
begin
  assign(f, 'goto.in');
  reset(f);
  cgoto := 0;
  state := prog;
  while not eof(f) do
  begin
    readln(f, s);
    for i := 1 to length(s) do
    case state of
      com: if s[i] = '}'  then state := prog;
      str: if s[i] = '''' then state := prog;
      prog:
        if s[i] = '''' then state := str else
        if s[i] = '{'  then state := com else
        if (s[i] in ['g', 'G']) and
          ((i = 1) or (s[i - 1] in sr)) then
         begin
           s1 := copy(s, i, 5);
           for j := 1 to length(s1) do
             s1[j] := upcase(s1[j]);
           if (s1 = 'GOTO') or (s1 = 'GOTO ')
           or (s1 = 'GOTO{') then inc(cgoto)
         end
    end {case}
  end;  {while}
  assign(g, 'goto.out');
  rewrite(g);
  writeln(g, cgoto);
  close(g)
end.
```

9. В этой задаче отрабатывается умение слияния двух упорядоченных последовательностей. Задача должна решаться за время, пропорциональное чтению каждого из файлов:

```
var f1, f2, f3: text;
  a, b: integer;
begin
  assign(f1, 'f1.txt');
  assign(f2, 'f2.txt');
  assign(f3, 'f3.txt');
```

```
  rewrite(f3);
  reset(f1);
  reset(f2);
{подразумевается, что хотя бы по одному числу
   в каждом из исходных файлов есть}
  read(f1, a);
  read(f2, b);
  if a < b then write(f3, a) else write(f3, b);
  while (a < b) and not eof(f1) or
        (b <= a) and not eof(f2) do
  begin
    if a < b then  read(f1, a) else  read(f2, b);
    if a < b then write(f3, a) else write(f3, b)
  end;
{один из файлов исчерпан}
  while not eof(f2) do
  begin
    write(f3,b);
    read(f2,b)
  end;
  while not eof(f1) do
  begin
    write(f3, a);
     read(f1, a)
  end;
  if a > b then write(f3, a) else write(f3, b);
  close(f3)
end.
```

Урок 16

Задачи

1. Первый способ основан на том факте, что симметрическая разность состоит из элементов, содержащихся в объединении двух множеств, за исключением их пересечения:

```
s := s1 + s2 - s1 * s2;
```

Второй способ выглядит так:

```
s := (s1 - s2) + (s2 - s1);
```

2. Приведем фрагменты программы, решающие поставленные подзадачи:

```
if T * T1 = [] then T := T + T1
...
if R1 - R <> [] then
begin
  for  i := rmin to rmax do
    if  i in (R1 - R) then
    begin
      R := R + [i];
      break
    end
end;
```

5. Решение этой задачи похоже на генерацию сочетаний (см. задачу 7 урока 14). Приведем основную процедуру:

```
procedure Permutations(n: integer);
var s: set of byte;
 procedure Perm(i: integer);
 var j, k: integer;
 begin
   if i > n then print(p, n)
   else {заполняем i-е место в перестановке}
     for j := 1 to n do
       if not (j in s) then
       begin
         p[i] := j;
         s := s + [j];
         Perm(i + 1);
         s := s - [j]
       end
 end; {Perm}
begin {Permutations}
  s := [];
  Perm(1)
end;  {Permutations}
```

Магазин «Математическая книга»

Книги издательства МЦНМО можно приобрести в магазине «Математическая книга» в Москве по адресу: Б. Власьевский пер., д. 11; тел. (499) 241-72-85; biblio.mccme.ru

Книга — почтой: http://biblio.mccme.ru/shop/order

Книги в электронном виде: http://www.litres.ru/mcnmo/

Мы сотрудничаем с интернет-магазинами

- Книготорговая компания «Абрис»; тел. (495) 229-67-59, (812) 327-04-50; www.umlit.ru, www.textbook.ru, абрис.рф
- Интернет-магазин «Книга.ру»; тел. (495) 744-09-09; www.kniga.ru

Наши партнеры в Москве и Подмосковье

- Московский Дом Книги и его филиалы (работает интернет-магазин); тел. (495) 789-35-91; www.mdk-arbat.ru
- Магазин «Молодая Гвардия» (работает интернет-магазин): ул. Б. Полянка, д. 28; тел. (499) 238-50-01, (495) 780-33-70; www.bookmg.ru
- Магазин «Библио-Глобус» (работает интернет-магазин): ул. Мясницкая, д. 6/3, стр. 1; тел. (495) 781-19-00; www.biblio-globus.ru
- Спорткомплекс «Олимпийский», 5-й этаж, точка 62; тел. (903) 970-34-46
- Сеть киосков «Аргумент» в МГУ; тел. (495) 939-21-76, (495) 939-22-06; www.arg.ru
- Сеть магазинов «Мир школьника» (работает интернет-магазин); тел. (495) 715-31-36, (495) 715-59-63, (499) 182-67-07, (499) 179-57-17; www.uchebnik.com
- Сеть магазинов «Шаг к пятерке»; тел. (495) 728-33-09, (495) 346-00-10; www.shkolkniga.ru
- Издательская группа URSS, Нахимовский проспект, д. 56, Выставочный зал «Науку — Всем», тел. (499) 724-25-45, www.urss.ru
- Книжный магазин издательского дома «Интеллект» в г. Долгопрудный: МФТИ (новый корпус); тел. (495) 408-73-55

Наши партнеры в Санкт-Петербурге

- Санкт-Петербургский Дом книги: Невский пр-т, д. 62; тел. (812) 314-58-88
- Магазин «Мир науки и медицины»: Литейный пр-т, д. 64; тел. (812) 273-50-12
- Магазин «Новая техническая книга»: Измайловский пр-т, д. 29; тел. (812) 251-41-10
- Информационно-книготорговый центр «Академическая литература»: Васильевский остров, Менделеевская линия, д. 5
- Киоск в здании физического факультета СПбГУ в Петергофе; тел. (812) 328-96-91, (812) 329-24-70, (812) 329-24-71
- Издательство «Петроглиф»: Фарфоровская, 18, к. 1; тел. (812) 560-05-98, (812) 943-80-76; k_i_@bk.ru, k_i_@petroglyph.ru
- Сеть магазинов «Учебная литература»; тел. (812) 746-82-42, тел. (812) 764-94-88, тел. (812) 235-73-88 (доб. 223)

Наши партнеры в Челябинске

- Магазин «Библио-Глобус», ул. Молдавская, д. 16, www.biblio-globus.ru

Наши партнеры в Украине

- Александр Елисаветский. Рассылка книг наложенным платежом по Украине: тел. 067-136-37-35; df-al-el@bk.ru